THE THREAT HUNTER'S COOKBOOK

SURGe by Splunk

Dr. Ryan Fetterman and Sydney Marrone
Featuring a foreword by Ryan Kovar

splunk> a CISCO company

Dr. Ryan Fetterman
Senior Manager, SURGe

Ryan is Senior Manager of Security Research on Splunk's SURGe team, joining after a decade spent in windowless basements conducting government research and consulting. He is a co-author of the PEAK Threat Hunting Framework and a contributor to various open-source security efforts. Ryan holds Doctorate and Masters degrees in engineering from George Washington University and a cybersecurity undergraduate degree from Penn State University. His favorite part of any project is when someone says "We just got new data."

Sydney Marrone
Principal Threat Hunter, Splunk

Sydney Marrone is a Principal Threat Hunter at Splunk and co-author of the PEAK Threat Hunting Framework. A proud thrunter, she is dedicated to advancing the craft of threat hunting through hands-on research, open-source collaboration, and community-driven initiatives. With a passion for making security knowledge accessible and actionable, she creates resources, leads workshops, and shares insights that inspire curiosity and empower defenders. When not hunting threats, she's lifting weights and keeping the hacker spirit alive.

FOREWORD

I love bread. Almost more than anything else. Definitely more than Marcus LaFerrera, the leader of SURGe, who asked me to write this foreword. There's just something magical about a boule of carbohydrates fresh out of the oven, its crust cracking as it cools, or the soft miga of pan de horno soaking up the juices from chorizo a la sidra — bread makes life better.

But it's not just about taste. Bread is about simplicity: water, yeast, flour, and salt. Four humble ingredients that can become a French baguette, focaccia, challah, or a dinner roll, all depending on how you mix and augment them. Baking, when you boil it down (ha!), is just chemistry: activate the rising agent, stretch the gluten, let steam create lift. OK, hold that thought — I need to run to the boulangerie.

So, after reading this book, I realized that threat hunting is kind of like baking bread. Especially when you're new to it. At first, it feels intimidating and overly complicated. But really, it all boils down to a few simple ingredients, or in the case of threat hunting, questions like these:

- Are you looking for a **deviation**?
- Are you working with **new data**?
- Are you searching for **commonalities**?
- Are you hunting for a **known indicator**?

When I first started threat hunting 15 years ago, there was no cookbook — because the practice of "threat hunting" wasn't defined like it is today. It was hard to figure out what our "flour" and "water" were. How long do you mix a stats command? When do you take your loadjob out of the oven? There weren't any recipes to help guide the process.

That's why I'm jealous of you. This book is the resource I wish I'd had 15 years ago. It's a collection of recipes for threat hunting that gives you a clear problem statement, a step-by-step solution, and even resources to go deeper into each problem area. It's like learning to bake: follow the recipe, check the results (taste the bread), jot down your notes (update the recipe), and try again (classic OODA loop).

And like the best bakers, once you've nailed the basics, it's time to experiment. Explore new data, tackle different problems, and don't be afraid to get creative. Remember, someone once looked at stale cheese and thought, "Why not mix this into bread dough?" — and that's how we got pão de queijo, the cheesy bread you devour at Brazilian steakhouses.

So, roll up your sleeves, grab some SPL, and start hunting. And hey, if it ever feels like too much, just remember: whether it's bread or threats, it's all about how you mix the ingredients.

Sincerely yours,

Ryan Kovar
The Baker
Former Splunker and member of SURGe
Currently a Cyber Raconteur

CONTENTS

ALLEZ CUISINE

Welcome threat hunters, intel analysts, Splunk enthusiasts, data scientists, and students. Today, we are all chefs — "cyber-chefs," that is — exploring the discipline of threat hunting through the structure of a cookbook.

Like cooking, threat hunting is equal parts art and science. While our primary focus as threat hunters is to methodically and proactively find security incidents our defenses have otherwise missed, there are always multiple paths available for us to take. Balancing control with creativity impacts both the outcome of our efforts and the quality of the results. The challenge as a hunter is to develop the awareness and expertise to choose the best approach.

This cookbook is designed to help you learn and hone those skills. It's specifically crafted to expand on your SPL (Splunk's Search Processing Language) know-how by filling the gap between the theory of the PEAK Threat Hunting Framework and the expansive functionality of Splunk.

From our kitchen to yours

The SURGe team at Splunk considers threat hunting to be the driver to improving your overall security program. Through this process, you may discover unknown security incidents, identify misconfigurations and vulnerabilities in your environment, pinpoint gaps in your data visibility, and improve organizational communication and understanding of your network environment.

For more detail, check out the PEAK Threat Hunting Framework, a practical, vendor-agnostic, customizable approach to threat hunting, designed to help organizations create or refine their threat hunting programs.

While PEAK outlines the broader threat hunting processes you can follow, the following sections of this book will explore the hunting methods you can employ, one at a time. Methods are the top-level categories of data analysis approaches to use when conducting your hunt. Fundamental methods include, for example, sorting, stacking, clustering, and grouping. Within these sections, recipes are provided, containing specific example implementations within each method. The final sections will introduce more advanced topics, like applying machine learning techniques, pre-processing data for visualization, combining recipes, and implementing useful Splunk add-ons for cybersecurity data.

This e-book is a useful practitioner's reference in that all bolded SPL commands are linked to their documentation page and each section contains additional Splunk resources for deeper exploration of the topics. If you're new to hunting, here is some recommended pre-reading to better understand the process and the motivations of threat hunting:

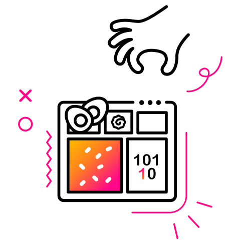
- **What Is Threat Hunting? A Cybersecurity Strategy >**
- **How to Start Threat Hunting: The Beginner's Guide >**

How to use this book

There are a number of ways to approach how you read and use this book. It's all, after all, a matter of taste. Here are just a few options:

- If you've started a hunt, try reading the sections describing each method and consider how the ideas could fit your hunt topic, or check the "Choosing the right recipe" section to follow guidance to the right approach.
- Jump right to the examples, using or adapting the queries and ideas like building blocks for your own hunt.
- Kick up your feet and read it end to end. Who knows — you might learn something!

These examples often merely showcase what is possible. Chances are, you'll have to slightly modify these queries to match the fields and values in your own environment.



MISE EN PLACE

(BUT FOR THREAT HUNTING)

Choosing the right recipe

We begin with the age-old question: "So, what's for dinner?" When you first enter the kitchen, it can be overwhelming to figure out what to cook. One approach is to start by checking which ingredients (or in this case, data) you have available. If this information is not yet cataloged, you may want to start with a baselining approach, which is designed to help you profile and understand your data and its contents.

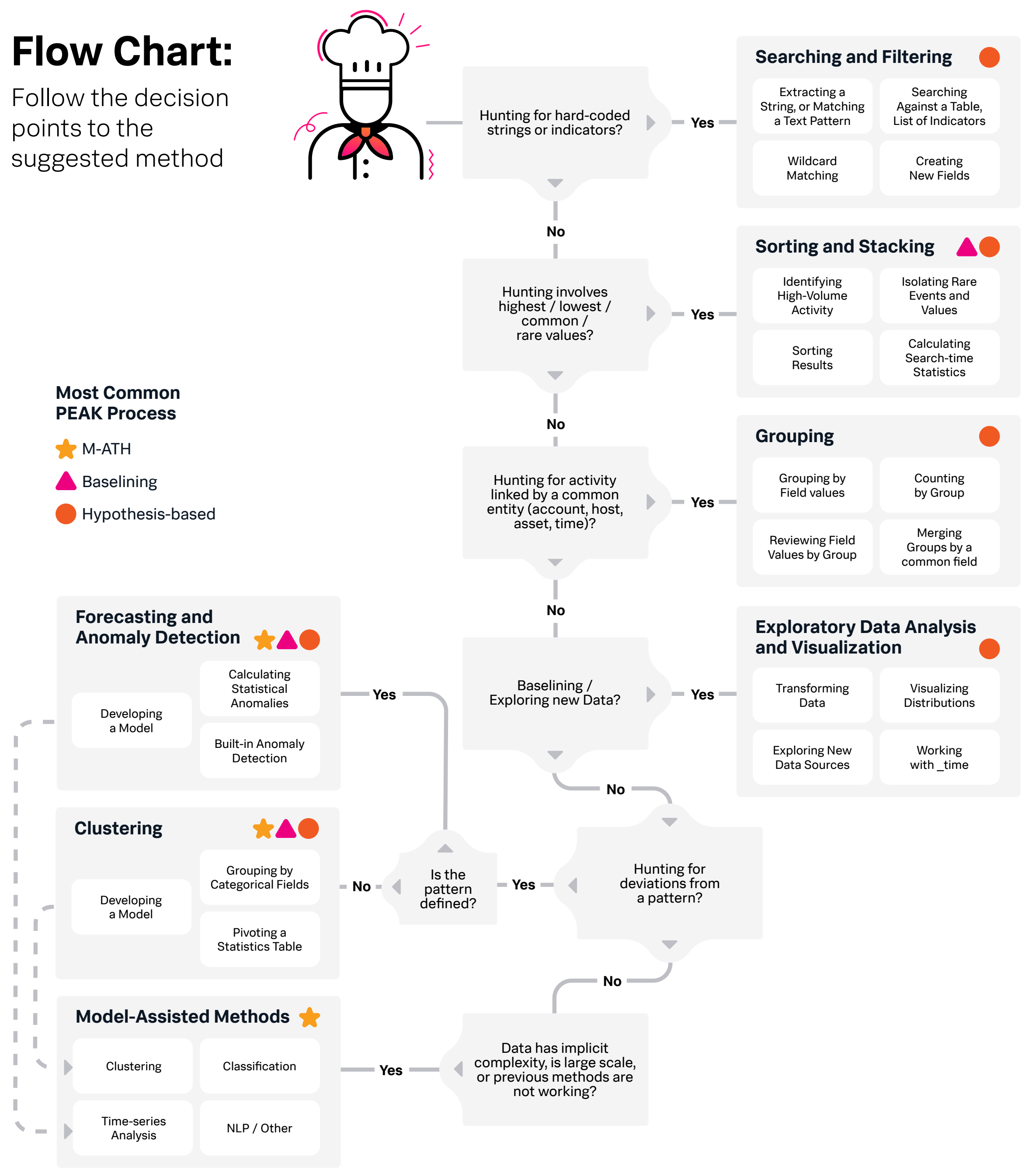
Once you're familiar with the data you have available, next up is selecting a topic. Your topic is the focus of your hunt, the "what" you are trying to find, and the method will be "how" you search. Your topic can be inspired by recent threat reporting, hypotheses about how adversaries might target your high-value assets, or exploration of adversary behaviors in the MITRE ATT&CK® techniques. If you're looking for an idea to build on, check out the hunting queries from the Splunk Threat Research Team.

Choosing what method you use to hunt your topic is where the creativity begins. Use the flow chart here as a guideline to help get you thinking. If you're following PEAK, you may also notice which methods are common for baselining, model-assisted, or hypothesis-based hunting.

Tighten those apron strings and roll up your sleeves. This is where the real fun begins. In the subsequent pages, you'll find explainers of specific hunting methods, including when you should consider using the method and what exactly each one entails. **Your adventure begins, as it so often does, with a search.**

Flow Chart:

Follow the decision points to the suggested method


Hunting for hard-coded strings or indicators?
Yes
No
Searching and Filtering ●
Extracting a String, or Matching a Text Pattern
Searching Against a Table, List of Indicators
Wildcard Matching
Creating New Fields
Hunting involves highest / lowest / common / rare values?
Sorting and Stacking ▲●
Identifying High-Volume Activity
Isolating Rare Events and Values
Sorting Results
Calculating Search-time Statistics
Hunting for activity linked by a common entity (account, host, asset, time)?
Grouping ●
Grouping by Field values
Counting by Group
Reviewing Field Values by Group
Merging Groups by a common field
Baselining / Exploring new Data?
Exploratory Data Analysis and Visualization ●
Transforming Data
Visualizing Distributions
Exploring New Data Sources
Working with _time
Hunting for deviations from a pattern?
Is the pattern defined?
Forecasting and Anomaly Detection ★▲●
Developing a Model
Calculating Statistical Anomalies
Built-in Anomaly Detection
Clustering ★▲●
Developing a Model
Grouping by Categorical Fields
Pivoting a Statistics Table
Data has implicit complexity, is large scale, or previous methods are not working?
Model-Assisted Methods ★
Clustering
Classification
Time-series Analysis
NLP / Other
Most Common PEAK Process
★ M-ATH
▲ Baselining
● Hypothesis-based

SEARCHING AND FILTERING

Summary	SPL Functions	
Your go-to, your Chef's knife, Searching and Filtering is for slicing, dicing, and extracting the data you need.	• metadata • datamodel • xyseries • untable • eval	• logical operators • relational operators • lookup • regex • rex

When to use this method

You could employ searching and filtering when you're hunting for hard-coded strings or indicators.

You'll often use this method to:

- Search against an indicator list, via lookup.
- Pattern match, or extract precise strings, with regex and rex.
- Explore the contents of the Splunk® platform with metadata and datamodel.
- Write a lightning-fast accelerated search with tstats.

What this method entails

The simplest method of hunting, **searching** and **filtering** are the processes of querying data for specific artifacts like strings, event codes, or combinations of conditions from your field-value pairs. This is usually the starting point for developing a query: We first search and filter to our broadest, but most specific, set of underlying data that will contain the type of activity we are attempting to identify.

For example, if you're hunting logon events in Windows, you may start your search by identifying the index, sourcetype, and relevant Event IDs in your Windows logs.

When hunting, it can be useful to break the problem up into two parts: identification and classification. In a threat hunting workflow, the identification phase often starts with searching and filtering, rather than the search being purely a stand-alone method. During the identification phase, we want to create the most specific search possible (i.e., high precision), while still capturing all potential events of interest for the behavior we are targeting (i.e., high recall).

We can then apply the methods in later sections to help us with classification phase, like distinguishing between benign and malicious events in our dataset.

Jump to Recipes

Further reading

Write Better Searches >

Search Best Practices >

Using the Lookup Command for Threat Hunting (Lookup Before You Go-Go) >

Using Workflow Actions & OSINT for Threat Hunting in Splunk >

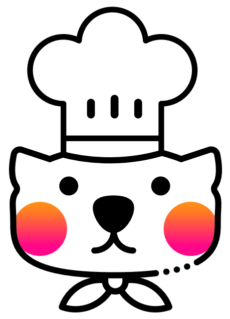
Stat! 3 Must-Have Data Filtering Techniques >

Using RegEx for Threat Hunting (It's Not Gibberish, We Promise!) >

Using eval to Calculate, Appraise, Classify, Estimate & Threat Hunt >

Detecting New Domains in Splunk >

Chef's Tips



- Time is a powerful filter, so be sure to select the smallest appropriate window. Do not use "all time" searches.
- Be as specific as possible! Specify your index, sourcetype, host, and so forth.
- Search by inclusion is better than search by exclusion.
- Consider the order of your search operation; filter as soon as possible.
- Minimize the use of wildcards such as *.

SORTING AND STACKING

Summary	SPL Functions	Algorithms and Visualizations
Interested in the highest volume or the rarest values? Try Sorting and Stacking to see what piles up or sifts through.	• `stats` • `top` • `rare` • `sort` • `eventstats` • `streamstats`	• `bar and column charts` • `line and area charts`

When to use this method

Consider a sorting and stacking method when your hunt involves highest, lowest, common, or rare values.

You'll often use this method to:

- Find rare processes or executables.
- Find highly active users, or high event volume.
- Assess high and low trends over time.

What this method entails

Also known as "stack counting" or "frequency analysis," **sorting** and **stacking** involves counting the number of occurrences for values of a particular type, considering, for example, the most and least common values of a particular field.

In practice, unusually high outbound web traffic from a host could be a sign of data exfiltration. An exceptionally rare executable or domain could be an indication of a compromised host. The key strength of sorting and stacking lies in its ability to reveal both high-frequency behaviors (indicative of mass exploitation techniques or widespread use of a particular tool) and low-frequency outliers (which might signal targeted attacks or novel threats).

Combining these sorting and stacking approaches with other techniques, such as filtering by time ranges, correlating results with known indicators of compromise (IOCs), or grouping multiple suspicious behaviors together (e.g., unusual login activity paired with rare process execution), can greatly increase the effectiveness of threat hunting.

Jump to Recipes

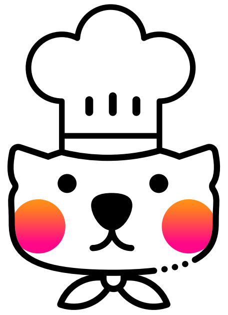
Further reading

Peeping Through the Windows (Logs): Using Sysmon & Event Codes for Threat Hunting >

Using stats, eventstats & streamstats for Threat Hunting…Stat! >

Detecting Lateral Movement with Splunk: How to Spot the Signs >

Chef's Tips



- You can stack across *multiple fields* with stats. Adding fields after your BY clause will perform your stats operation *per each unique combination of values.*
- If no count is specified when sorting, the default limit of 10000 is used. To return all results, specify `limit=0`.

GROUPING

Summary	SPL Functions	
Looking for complimentary flavors, like lateral movement + privilege escalation? Grouping will connect actions and events into groups with these functions.	• `bin` • `stats` – `values` – `dc` – `count, avg, min, max...`	• `join` • `transaction`

When to use this method

Employ grouping when you're hunting for an activity linked by a common entity, such as an account, host, asset, or time.

You'll often use this method to explore hypotheses about a predefined set of events:

- Grouping activity across adversarial tactics, like Initial Access → Privilege Escalation, or grouping system Discovery → Lateral Movement.
- Accessing a personal webmail account → Downloading spear phishing attachment.
- Connecting the use of specific techniques, like encoded command execution → malware download.

What this method entails

Grouping involves identifying instances where multiple unique artifacts, such as IP addresses, file hashes, domains, or user behaviors, appear together based on specific criteria. Unlike clustering, which often analyzes broader datasets and seeks to find natural groupings within the data, grouping is more focused and deliberate, working with a predefined set of items that are already flagged as potentially suspicious or of interest. The key idea is to group related artifacts that might indicate a common tactic, technique, or procedure (TTP) used by an attacker, often within a specific timeframe.

The groups formed through this method can reveal relationships between various elements of an attack, potentially pointing to a specific tool or technique. For example, a group might consist of multiple unusual login attempts, lateral movement between systems, and rare network connections — all occurring within the same time frame. Such a pattern could suggest an attacker moving within the network or attempting to escalate privileges.

Grouping can be particularly effective in revealing multi-stage attacks, where an adversary performs several different actions in succession (e.g., reconnaissance, exploitation, data exfiltration). This allows hunters to make use of what might otherwise be considered "weaker signals" of malicious activity, like Account Discovery, or Remote System Discovery actions, and coupling them with more significant potential indicators.

As you think about grouping and move onto the clustering and forecasting and anomaly detection methods, consider how attributes like volume and frequency relate to your hunt topic. Is the overall quantity (i.e., volume) of a specific activity interesting in relation to your topic? What about a threshold of volume with a specific timespan (i.e., frequency)?

Jump to Recipes

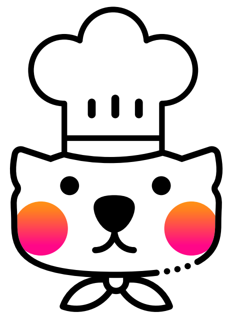
Further reading

Stat! 3 Must-Have Data Filtering Techniques >

Using stats, eventstats & streamstats for Threat Hunting...Stat! >

Search commands > stats, chart, and timechart >

Chef's Tips



- Working with time influences how you group events. When you are grouping or stacking activity, consider how you bucket your time, what timeframe you search across, and by what unit you aggregate, e.g.:

```
| bucket _time span=1m as minute
| eval minute=strftime(minute, "%Y-%m-%dT%H:%M:%S")
```

  Note: bucket is actually an alias of the bin command

- If you're doing risk-based alerting, try hunting in your risk index. The risk index is where you are already collecting events of interest; testing some grouping ideas may help you tune the correlation searches that are populating the risk index or help you create some new thresholds for triggering alerts.
- Just as you can stack multiple fields with stats, you can use the same method to group. For example, connecting named pipe creation and connection events with Sysmon:

```
index=sysmon source="xmlwineventlog:microsoft-windows-sysmon/operational" EventCode IN (17,18)
| stats count by PipeName host EventCode EventDescription process_name
```

FORECASTING AND ANOMALY DETECTION

Summary	SPL Functions	Algorithms and Visualizations
Searching for something out-of-the ordinary? Anomaly Detection recipes will help you find events that stand out.	• `fillnull` • `timechart` • `eventstats` • `eval` • `anomalydetection`	• `DensityFunction` • `OneClassSVM` • `ARIMA` • `StateSpace Forecast`

When to use this method

Forecasting and anomaly detection are useful when you're hunting for deviations from a pattern, and that pattern is defined.

Forecasting and anomaly detection are best suited for cases where:

1. You have a good sample of observable behavior from the past to develop a model.
2. You expect to forecast the perceived boundaries of "normal" behavior out into the future.
3. You score new incoming data against those boundaries to determine how severe of an outlier is appropriate. One percent outside of the boundary might be worth investigating at your next opportunity, and 3x the normal boundary probably means the kitchen is on fire!

Some hunts you might perform using forecasting and anomaly detection include:

- A spike in failed login activity, e.g., Brute Force.
- User behavior-based analytics, like accessing sensitive data or network shares.
- Uncommon data transfer volumes, indicating potential data exfiltration.

What this method entails

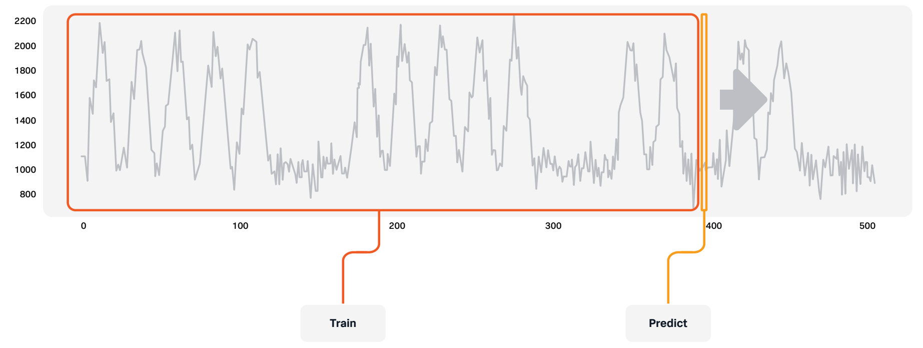
Statistics play a crucial role in analyzing data and identifying patterns, including detecting outliers that deviate significantly from the norm. However, in cybersecurity, data isn't always normally distributed, not all anomalies signify malicious activity, and sometimes, unusual data can be totally benign.

Keeping these principles in mind, anomalies are often still interesting for threat hunting and can be useful in the right context. Remember, the goal of threat hunting isn't solely to find malicious activity. Anomalies can be beneficial for helping us better understand our data, identify misconfigurations, or gain valuable insight into how users interact with resources across the network. So, in this section we'll cover methods that look for patterns, cycles, and deviations from "normal."

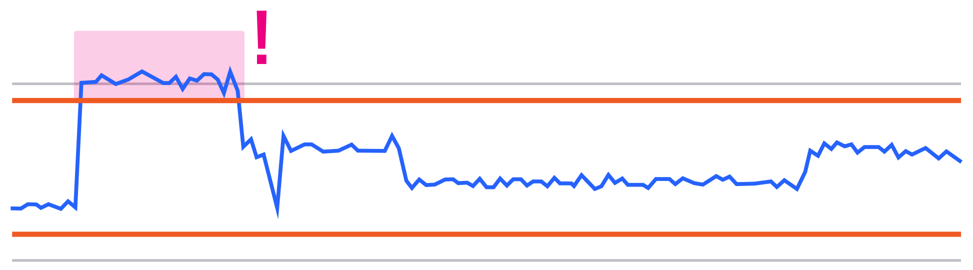
Forecasting is the process of predicting future trends, values, or events based on historical data. It is appropriate for threat hunting analysis when there is a time-based dimension to the data and patterns or trends from the past can reasonably be expected to inform future outcomes. This can include examples such as user pattern-of-life behavior, demand cycles in web-based services, or resource use of critical systems. Forecasting typically requires a window of time from which to learn the "normal" pattern of activity. This pattern can then be forecast ahead as a prediction, and anomalies can be identified in comparison to the actual events.

Anomaly detection is the identification of unusual or rare patterns in data that do not conform to expected behavior. For this reason, anomaly detection is often used in conjunction with forecasting. Since anomalies are not inherently malicious, they may often be more useful for threat hunting exercises rather than detection and alerting. If something is anomalous, it is interesting for one reason or another. And since the end goal of hunting is not just to identify incidents, but to expand our knowledge in ways that improve our defenses, these anomalies can be great sources for making important discoveries.

Forecasting example



Anomaly detection example

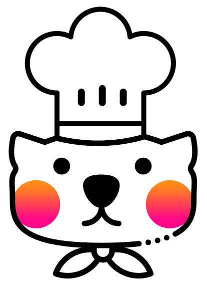


Jump to Recipes

Further reading

Data Downtime with Anomaly Detection >

Chef's Tips



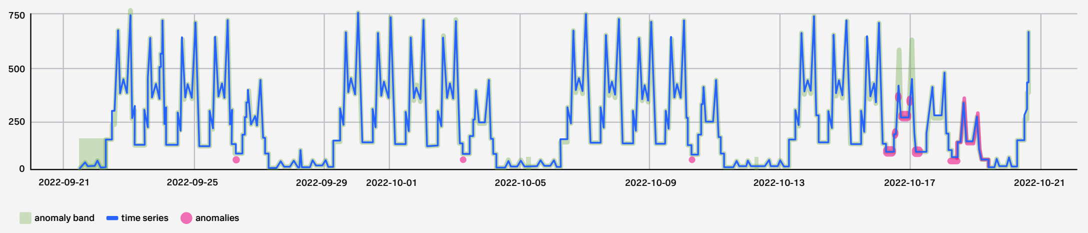
- Mean, median, and mode can provide key insights when baselining your data: the mean gives the average value; the median indicates the middle value when data is sorted; and the mode shows the most frequently occurring value. Together, they help identify the skewness and symmetry of the distribution, revealing whether data is evenly distributed or biased towards certain values.

mean	`| stats avg(field_name) as mean_value`
median	`| stats median(field_name) as median_value`
mode	`| stats count by field_name` `| sort -count` `| head 1` `| rename field_name as mode_value`

- A statistical outlier has defined thresholds, for example, 1.5 multiplied by the Interquartile Range (IQR). However, you can adjust the constants in these calculations higher or lower depending on the quality or amount of the results you are returning. You can decide where the line for anomalous falls — don't let science get in the way of the art!
- Are there nulls in your data? Consider how you should treat them. You may want to use the SPL fillnull command to remedy nulls with a default value. Or you might want a separate alert for the presence and frequency of nulls, as monitoring the cleanliness of the data that your model is digesting can help you monitor its health. Don't feed your model garbage!

- Forecasting over a long period? Consider your options for handling seasonality. Seasonality in time series analysis refers to regular, repeating patterns or cycles in data that occur at consistent intervals, such as daily, weekly, monthly, or yearly. Seasonal patterns can introduce predictable fluctuations in the data, and failing to account for these patterns can lead to inaccurate forecasts or incorrect identification of anomalies, mistaking normal seasonal variations for unusual or unexpected behavior.
- While an outlier is a statistical concept, an anomaly is context-dependent. Consider the alternatives here when selecting an approach, and when setting anomaly-criteria, based on the specific context and objectives of your hunt.

Anomalies among a seasonal pattern



CLUSTERING

Summary	SPL Functions	Algorithms and Visualizations
Wondering which ingredients would stick together? Use Clustering to automatically measure associations between events based on the attributes you define.	• `stats` • `cluster` • `fields` • `kmeans` • `xyseries`	• `K-Means` • `PCA` • `TFIDF` • `3D Scatter Plot`

When to use this method

Use a clustering technique when you're hunting for deviations from a pattern, and that pattern is not defined.

What this method entails

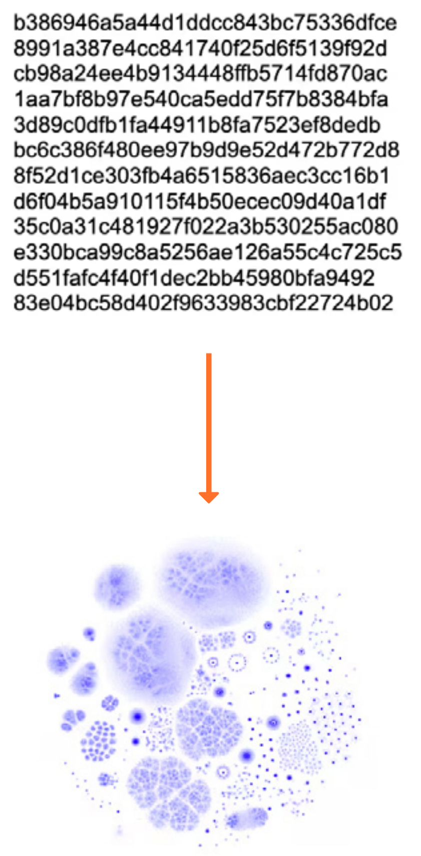
Clustering is an analysis technique often used to automatically group similar data points into clusters based on specified characteristics within a larger dataset. Unlike traditional grouping, where categories are predefined, clustering algorithms determine the groups, and analysts must interpret the clusters to understand their meaning. This method is considered an unsupervised machine learning technique because it does not rely on labeled data.

In threat hunting, clustering involves grouping similar events or data points to uncover patterns, anomalies, or outliers that may indicate malicious activity. This can include clustering based on event types and their frequency, identifying time-based patterns, or analyzing behavioral similarities, such as common attack vectors or user behaviors.

Clustering is often used in threat hunting when we expect our normal data to fall into orderly, logical groups. For example:

- Clustering the JA3 signatures to distinguish malicious vs. legitimate executables

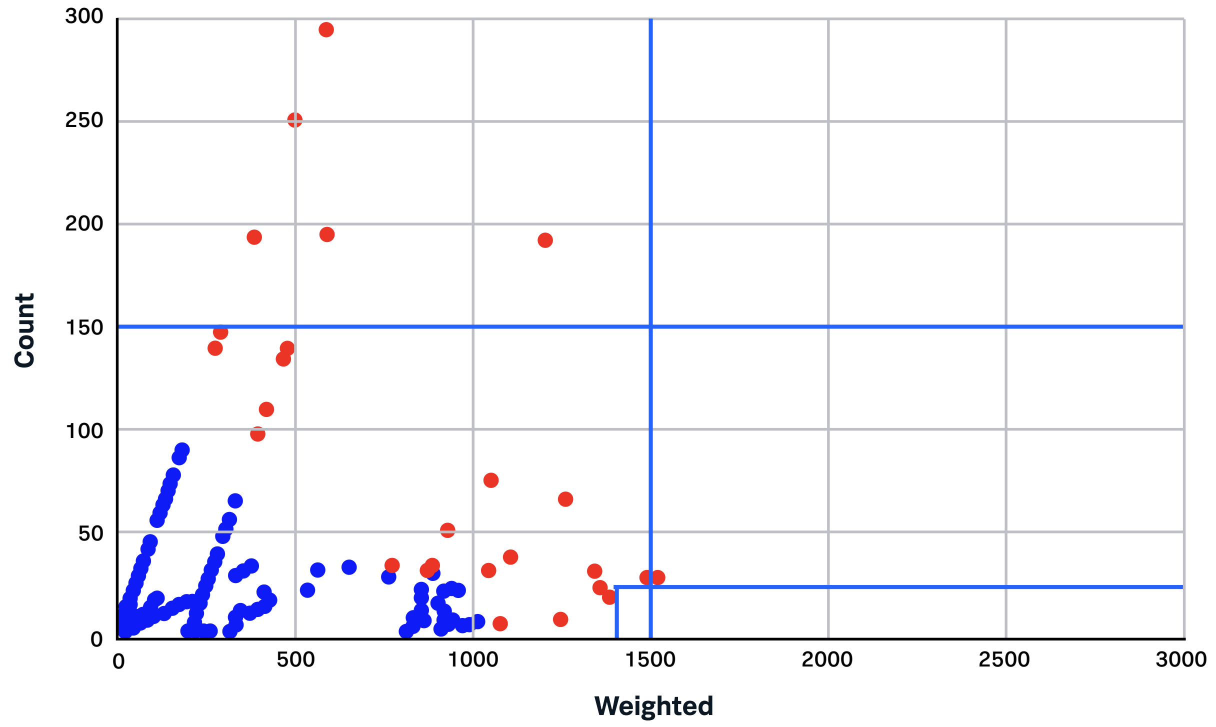
Clustering executables by attributes



```
b386946a5a44d1ddcc843bc75336dfce
8991a387e4cc841740f25d6f5139f92d
cb98a24ee4b9134448ffb5714fd870ac
1aa7bf8b97e540ca5edd75f7b8384bfa
3d89c0dfb1fa44911b8fa7523ef8dedb
bc6c386f480ee97b9d9e52d472b772d8
8f52d1ce303fb4a6515836aec3cc16b1
d6f04b5a910115f4b50ecec09d40a1df
35c0a31c481927f022a3b530255ac080
e330bca99c8a5256ae126a55c4c725c5
d551fafc4f40f1dec2bb45980bfa9492
83e04bc58d402f9633983cbf22724b02
```

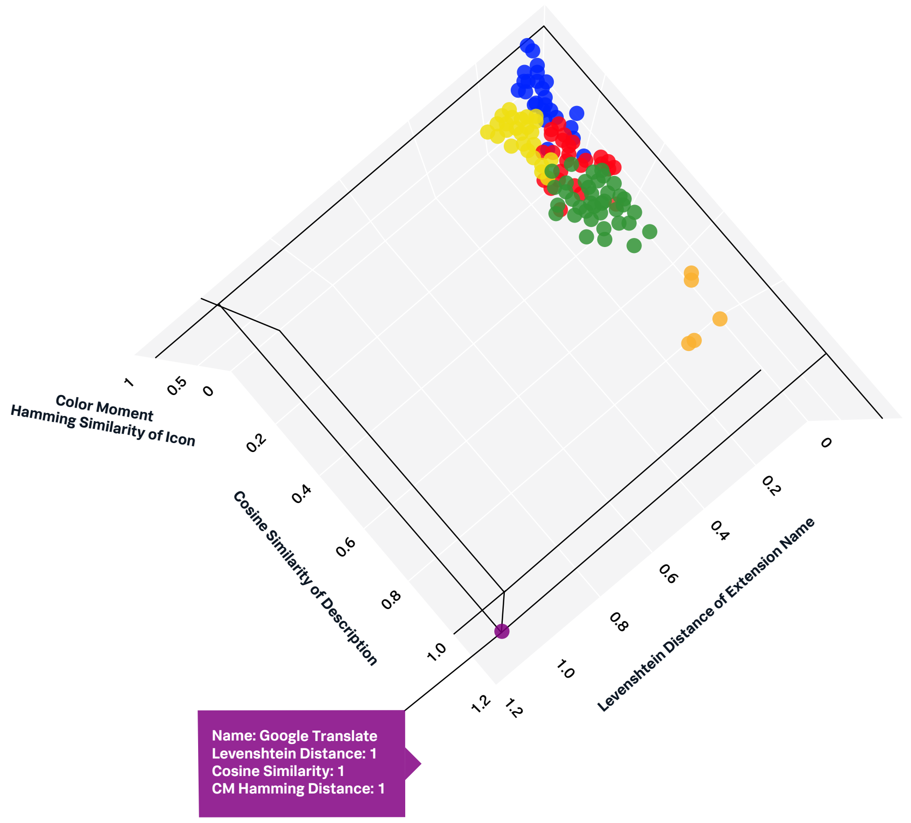
- Clustering encoded URI strings and user-agents

Clustering outliers via isolation forest



- Comparing feature similarity to hunt for masquerading browser extensions

Clustered data in a three-dimensional scatter plot



Jump to Recipes

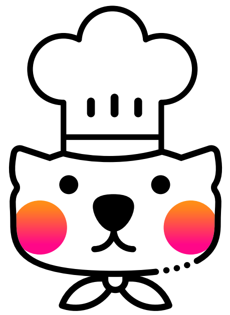
Further reading

Splunk > Clara-fication: transpose, xyseries, untable, and More >

Splunk Command > Cluster >

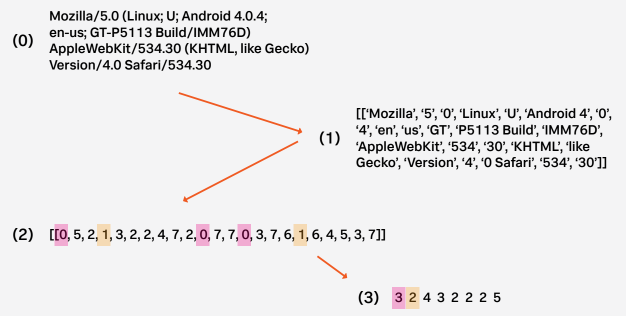
Analyzing Botnets with Suricata & Machine Learning >

Chef's Tips



- Use dimensionality reduction techniques like principal component analysis (PCA) to simplify high-dimensional data, making clustering more effective for identifying patterns in threat hunting.
- Cluster text-based fields by encoding them numerically, e.g., with term frequency-inverse document frequency (TFIDF). Representing strings as numbers allows you to measure distance or assess similarity between them as vectors.

Vectorizing user-agent strings



```
(0)  Mozilla/5.0 (Linux; U; Android 4.0.4;
     en-us; GT-P5113 Build/IMM76D)
     AppleWebKit/534.30 (KHTML, like Gecko)
     Version/4.0 Safari/534.30

(1)  [['Mozilla', '5', '0', 'Linux', 'U', 'Android 4', '0',
     '4', 'en', 'us', 'GT', 'P5113 Build', 'IMM76D',
     'AppleWebKit', '534', '30', 'KHTML', 'like
     Gecko', 'Version', '4', '0 Safari', '534', '30']]

(2)  [[0, 5, 2, 1, 3, 2, 2, 4, 7, 2, 0, 7, 7, 0, 3, 7, 6, 1, 6, 4, 5, 3, 7]]

(3)  3 2 4 3 2 2 2 5
```

EXPLORATORY DATA ANALYSIS AND VISUALIZATION

Summary	SPL Functions	Algorithms and Visualizations
Know thyself — but also, know thy data. EDA and Visualization techniques use statistics and graphical means to establish a baseline understanding of our data.	• `fieldsummary` • `transpose` • `chart` • `bin` • `metadata` • `timechart` • `sparkline`	• `Scatter Plot` • `Box Plot`

When to use this method

Exploratory data analysis (EDA) and **visualization** are foundational aspects of threat hunting. EDA is a wonderful catch-all term for the wide variety of analysis you can perform to figure out what comprises your data and what patterns exist within it. Executing these techniques properly will greatly assist your baseline hunt activities, as well as facilitate your data for developing hunting models or accurately testing hypotheses.

EDA and visualization is often associated with:

- Baselining threat hunts.
- Exploring outliers and trends.
- Understanding your data through graphs and distributions.

What this method entails

A thorough EDA typically involves calculating descriptive statistics (e.g., mean, median, mode), reviewing value distributions, and identifying patterns (e.g., through clustering, frequency analysis). These techniques may reveal underlying distributions, outliers, correlations, and trends that may be indicative of suspicious or malicious activity.

Visualization amplifies the effectiveness of EDA by turning raw data into intuitive, interpretable graphics. Graphical representations such as histograms, scatter plots, heat maps, and time-series charts enable threat hunters to quickly grasp complex data, or accurately communicate it to others! Techniques like heatmaps for access frequency, time-series visualizations for event occurrence patterns, and scatter plots for correlations between variables allow you to explore vast datasets more efficiently.

Jump to Recipes

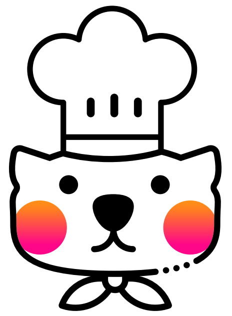
Further reading

Using metadata & tstats for Threat Hunting >

Dashboard Design: Visualization Choices and Configurations >

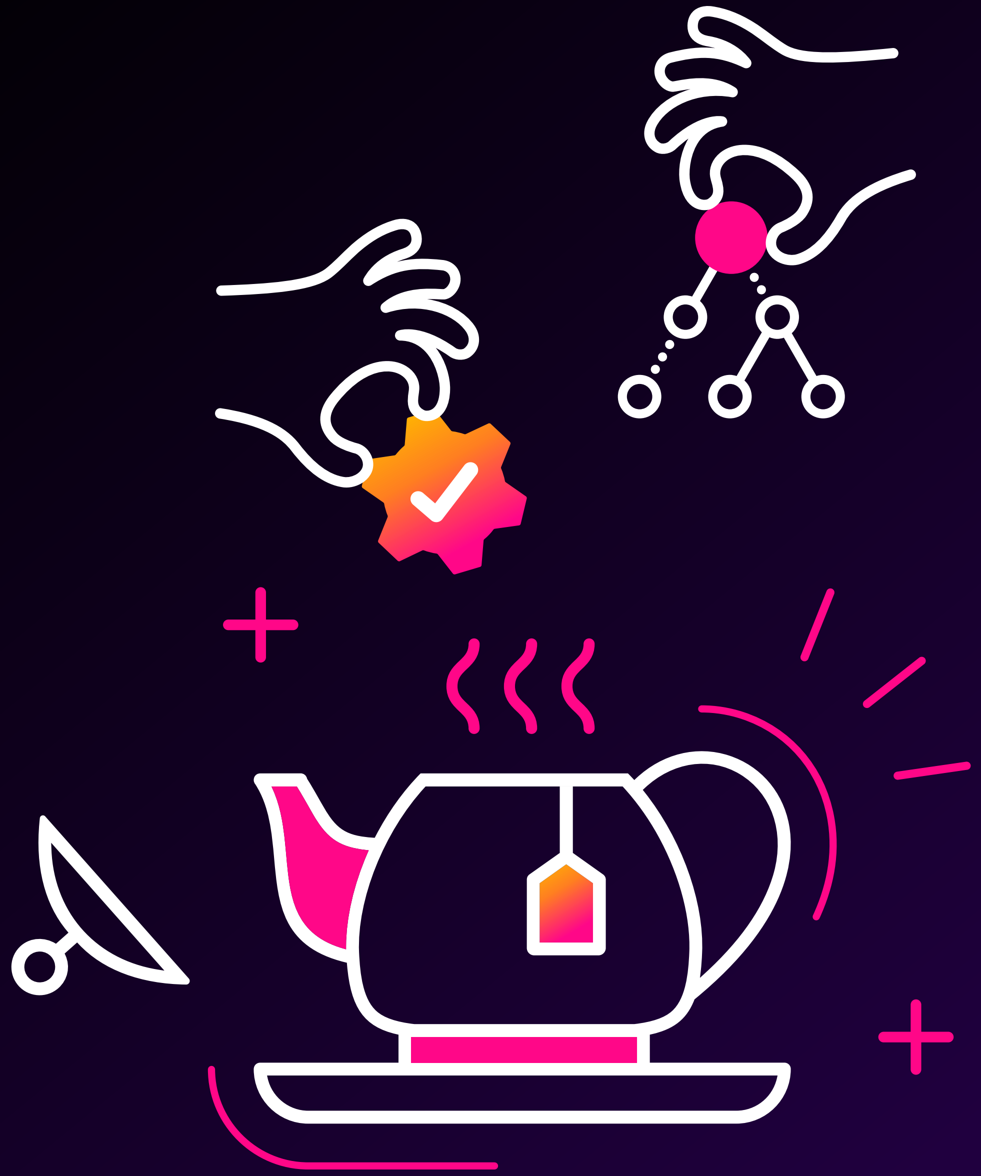
Exploratory Data Analysis for Anomaly Detection >

Chef's Tips



- Start with fieldsummary to shortcut your statistics calculation and quickly see an overview of your data source.
- By default, search events are listed in reverse chronological order (most recent first). In an incident investigation, you may want to reverse this, e.g.: | `sort -_time`, or invert current result order even faster with | `reverse` !

```
| stats count by PipeName host EventCode
EventDescription process_name
```



COMBINED METHODS

Summary	SPL Functions	Algorithms and Visualizations
A master class in culinary data science — apply these signature methods for testing multi-stage hypotheses, training classification models, or adding new ingredients to your repertoire!	• `iplocation` • `geostats` • `appendpipe` • `outputlookup`	• `DecisionTree Classifier` • `GradientBoosting Classifier` • `Logistic Regression` • `RandomForest Classifier` • `URL Toolbox` • `PSTree`

Now that we've covered the core methods for threat hunting, it's time to explore some combinations. Many of these recipes can and often are used together. For example, you may start by searching and filtering for relevant event data, exploring the dataset with exploratory data analysis (EDA) and clustering, before finally settling on a forecasting and anomaly detection method to finalize the results of your hunt.

The kitchen is open — apply these new tools as you see fit. In this section, we'll highlight some popular combinations, as well as some advanced examples.

Jump to Recipes

Further reading

Random Words on Entropy and DNS >

Parsing Domains with URL Toolbox >

Detecting Dubious Domains with Levenshtein, Shannon & URL Toolbox >

Tall Tales of Hunting TLS/SSL Certificates >

Detecting DNS Exfiltration with Splunk: Hunting your DNS Dragons >

Search commands for machine learning - Splunk Documentation >

Chef's Tips

- **To get started with advanced algorithms, you should first understand data science fundamentals like imputation (i.e., how to handle missing values), normalization (i.e., when and how to sample / balance your data), scaling (e.g., min-max, standardization), and data encoding (e.g., one-hot, label, ordinal). Don't be intimidated! Review some of the examples in this section to see how these concepts are applied to real-world problems.**
- **Before developing a model, ask yourself if this problem needs to use machine learning. Model-assisted threat hunting is the best selection only under certain conditions, i.e.:**
  - A simpler method will not have an equally effective result,
  - The data is complex and requires pre-processing or an advanced method, or
  - A machine-learning approach offers a novel opportunity to discover valuable insights.

THE RECIPES

Let’s get cooking

You should now have an understanding of the different categories of threat hunting methods, their functionality, and their applicability. We’ve included a reference chart at the end of the book with some of the topics and ideas we’ve covered.

If you need help thinking through your options, don’t forget to check the “Choosing the right recipe” chart.

- Your SPL knives are sharpened.
- You have a list of fresh ingredients.
- Time to dive in and cook up something wild.

Bon appétit, and happy hunting!

1. Searching and filtering

Filtering and combining search logic

Problem

You need to string together a combination of logical statements, including suspicious conditions, while excluding benign activity.

Solution

Complex logic can be built using combinations of field-value pairs and boolean logic.

- **Logical operators**
  - (AND [&&], OR [||], NOT [!=])
  - For example, filtering out trusted DNS sources with NOT:
    -
      ```
      sourcetype=stream:dns
      src=192.168.250.100
      NOT query IN(wpad*, isatap*,
      *.windows.com, *live.com)
       | stats count by query
      ```
- **Relational operators**
  - equals (=) or (==)
  - does not equal (!=)
  - is greater than (>)
  - is greater than or equal to (>=)
  - is less than (<)
  - is less than or equal to (<=)
    - For example, linking operators to searching for potential exfiltration conditions:
      -
        ```
        index=proxy dest_domain !=
        "trustedcompany.com" AND bytes_
        out > 1000000 AND response_time
        >= 5000 AND url_category !=
        "internal" AND status_code = 200
        AND bytes_in <= 50000 AND http_
        method == "POST"
        ```
- **IN** operator
  - The IN operator matches the values in a field to any of the items in the <expression-list>. The items in the <expression-list> must be a comma-separated list.
    - For example, looking for error events in HTTP traffic:
      -
        ```
        status IN("400", "401",
        "403", "404")
        ```

Discussion

Operators can be combined to create complex logic, including the ability to use parentheses to group how the logic is evaluated, e.g.:

```
index=security (sourcetype=firewall OR
sourcetype=proxy) AND (action=blocked
OR action=denied)
```

The parentheses ensure that the system first evaluates (`sourcetype=firewall OR sourcetype=proxy`) and then separately evaluates (`action=blocked OR action=denied`) before applying the AND condition between them.

As far as operators that are considered exclusion operators, both != and `NOT field=value` fall into that category, but they behave differently. The difference is that != implies that the field exists but does not have the value specified. So if the field is not found at all in the event, the search will not match.

`NOT field=value` on the other hand will check if the field has the specified value, and if it doesn't for whatever reason, it will match.

Extracting a string or matching a text pattern

Problem

You need to filter log events for error messages, but an error field is not extracted.

Solution

This query searches the `system_logs` dataset and uses the `regex` command to filter events where the `_raw` field contains either `error` or `fail`.

```
index=system_logs
| regex _raw="(?i)error|fail"
```

Discussion

`regex` offers incredible flexibility for advanced search and pattern matching. The `(?i)` flag ensures that the search is case insensitive and matches on both uppercase and lowercase versions of the words error and fail. There are many online resources for regex cheat sheets and sandboxes to experiment with the full scope of this utility.

```
regex (<field>=<regex-expression> |
<field>!=<regex-expression> |
<regex-expression>)
```

Problem

You want to narrow down events to a single network range, e.g. 192.168.224.0 — 192.168.225.255.

Solution

This query looks at the `ids` sourcetype and uses the `regex` command to filter for events where the `src_ip` field matches the specific pattern 192.168.224.x or 192.168.225.x, excluding events that don't fit this range.

```
index=network sourcetype=ids
| regex src_ip="192\.168\.(224|225)\
.\d{1,3}"
```

Discussion

`regex` applies regular expressions to filter events. It is a powerful tool for pattern matching. When used, it displays results that match the specified pattern. You can specify that the regex command applies to a particular field by using `<field>=<regex-expression>`; otherwise, it defaults to the `_raw` field.

```
regex (<field>=<regex-expression> |
<field>!=<regex-expression> | <regex-
expression>)
```

Problem

You need to extract domain names from the results of known phishing email addresses.

Solution

This query extracts the domain (e.g., @splunk.com) from the sender field in email events.

```
index=mail
| rex field=sender
"(?<domain>@[\w+\.\w+]+)"
```

Discussion

The key components of this regex pattern include:

- `@`: This matches the @ symbol, ensuring that the domain is extracted from an email address.
- `\w+`: This matches one or more word characters (letters, digits, and underscores), capturing the initial part of the domain name.

- `\.`: This matches the period separating different parts of the domain, like "splunk" and "com."
- `[\w+\.\w+]+`: This part ensures that multiple domain segments are captured, accommodating domains with subdomains (e.g., `@mail.splunk.com`).

This **rex** command assumes a basic domain structure and may need adjustments for more complex domain formats (e.g., those with additional subdomains or country-specific top-level domains like .co.uk).

Problem

You want to review the contents of a `passwd` field to ensure there are no unencrypted passwords being sent across the wire, but the field is not extracted from the raw event.

Solution

Use the `rex` command to extract the field during search time, using pattern matching. This query extracts the password from the `form_data` field where it has passed the `passwd=` parameter, saving the password to the `pass` field.

```
index=network sourcetype=stream:http
| rex field=form_data
"passwd=([?<pass>[^&+)"
```

Discussion

rex extracts field values using regular expressions and saves the values in a new assigned field. It is particularly useful for extracting structured data from unstructured text. Similar to regex, you can specify a particular field; otherwise, it defaults to the _raw field.

```
rex [field=<field>] ( <regex-
expression> [max_match=<int>] [offset_
field=<string>] ) | (mode=sed <sed-
expression>)
```

Problem

You need to parse JSON but the results include long, hard-to-read field names, or the values are nested and have become multi-valued in a single field, requiring additional processing to clean up.

For example, some data sources are presented as lists of dictionaries:

```
{
  "groupId": "sg-bb958cd3",
  "ipPermissionsEgress": [
    {
      "fromPort": 80,
      "ipProtocol": "tcp",
      "ipv4Ranges": [
        {
          "cidrIp": "0.0.0.0/0",
          "description": "Web"
        }
      ]
    },
    {
      "fromPort": 22,
      "ipProtocol": "tcp",
      "ipv4Ranges": [
        {
          "cidrIp": "192.168.1.123/32"
        }
      ]
    }
  ]
}
```

Solution

Use `spath` to navigate to the relevant parent list or dictionary. Include `fromjson` to automatically parse the JSON key-value pairs into simple extracted fields, reducing the need for manual manipulation.

```
index=aws-config sourcetype=aws:config
(source=*ConfigSnapshot* OR
source=*ConfigHistory*) _index_
earliest=-48h earliest=0 asset_
type=*EC2*
TERM(arn:aws:ec2:us-east-
2:xxxxx:security-group/sg-xxxxx)
| spath path=configuration.
ipPermissionsEgress{} output=foo
| table index foo
| mvexpand foo
| fromjson foo
| fields - foo
```

Discussion

The Splunk® platform often returns multi-values for keys nested inside a list of dictionaries. It can be difficult to join the entries together for each dictionary. A common mistake is using `mvexpand` on each key, which creates incorrect associations between the keys.

Instead of expanding each value incorrectly with `mvexpand`, it's better to parse the data properly. The `fromjson` command can automatically parse key-value pairs inside a JSON dictionary and convert them into events, simplifying the filtering process.

Wildcard-matching strings

Problem

You need to match variations of a string that begin with the same characters. For example, admin accounts are named d_admin1, d_admin2, d_admin3.

Solution

Filter events where the `username` starts with "admin" by using the `like` function with the % character as a wildcard to match any characters that follow. The `like` function supports SQL-style pattern matching with special characters like % for multiple characters and _ for single characters. It offers more control compared to a simple * wildcard.

```
host="host1337"
| where like(username, "%admin_")
```

Problem

You need to match IP addresses or a subnet, such as `10.9.165.*`.

Solution

The following query returns events from the host `host1337` by checking if the source IP starts with `10.9.165.` using the `like` function. It also applies `cidrmatch` to include events where the destination IP falls within a specific internal network range.

```
host="host1337"
| where like(src, "10.9.165.%") OR
cidrmatch("10.9.165.0/25", dest)
```

Solution (alternate)

Using the TERM keyword ensures that the search looks for an exact term, which in this case is the beginning of an IP address string.

```
index=host TERM("10.9.165.")
```

Discussion

When data is indexed, characters such as periods (.) and underscores (_) are recognized as minor breakers between terms. Use the TERM directive to ignore the minor breakers and match whatever is inside the parentheses as a single term. For example, the IP address 127.0.0.1 contains the period (.) minor breaker. If you search for the IP address 127.0.0.1, the Splunk platform searches for 127 AND 0 AND 1 and returns events that contain those numbers anywhere in the event. If you specify TERM(127.0.0.1), the search treats the IP address as a single term, instead of individual numbers, and returns all events that contain the IP address 127.0.0.1.

Problem

You want to identify web traffic using Firefox or Chrome browsers but can't match all the specific versions of user-agent strings.

Solution

Use the **eval** command with the case function to create a new field called browser_type, checking browsers based on the user_agent field. The like function checks for Chrome or Firefox in the user_agent, and if neither is found, the case function assigns the value to Other.

```
| eval browser_type=case(like(user_
agent, "%Chrome%"), "Chrome",
like(user_agent, "%Firefox%"),
"Firefox", 1==1, "Other")
```

Discussion

The **like** function can be used with the **eval**, **fieldformat**, and **where** commands and as part of eval expressions. The percent (%) symbol is the wildcard that you use with the like function. The following syntax is supported:

```
...| eval new_field=if(like(<str>,
<pattern>)
...| where like(<str>, <pattern>)
...| where <str> LIKE <pattern>
```

Searching against a table or a list of indicators

Problem

You are analyzing firewall data for suspicious Command and Control (C2) activity, but you want to exclude a list of good domains associated with your organization (e.g., internal domains, trusted partners) to focus on suspicious or unknown domains.

Solution

This query searches firewall events to compare the domains against a list of known good domains from a lookup file. If a match is found, it outputs the is_good value from the lookup, filters out events where is_good is NULL (which indicates unknown or suspicious domains), and then groups and counts the results by source.

```
index=firewall
| lookup known_good_domains.csv domain
AS domain OUTPUT is_good
| where isnull(is_good)
| stats count by src
```

Discussion

lookup and **inputlookup** allow you to work with CSV files in the Splunk platform. The lookup command is used to match data in the Splunk platform with data from a CSV or lookup table, adding context to the existing data. The inputlookup command retrieves the contents of a CSV or lookup file, allowing you to search the data within the lookup file itself.

When using the lookup command, if an OUTPUT or OUTPUTNEW clause is not specified, all of the fields in the lookup table that are not the match fields are used as output fields. If the OUTPUT clause is specified, the output lookup fields overwrite existing fields. If the OUTPUTNEW clause is specified, the lookup is not performed for events in which the output fields already exist.

```
| lookup <lookup-dataset> (<lookup-
field> [AS <event-field>] )...[ (OUTPUT
| OUTPUTNEW) ( <lookup-destfield> [AS
<event-destfield>] )...]
```

```
| inputlookup [append=<bool>]
[strict=<bool>] [start=<int>] [max=<int>]
<filename> | <tablename> [WHERE
<search-query>]
```

Optimizing searches

Problem

You need to investigate suspicious process executions across your Windows endpoints, but process data is spread across multiple sources.

Solution

Search the Processes dataset within the `Endpoint` data model, looking for events where the process name is either powershell.exe, wmic.exe, or cmd.exe. The search counts the occurrences of each of these processes by host and then groups the results by host and process name.

```
| datamodel Endpoint Processes search
| search Processes.process_name IN
("powershell.exe", "wmic.exe", "cmd.exe")
| stats count by host, Processes.
process_name
```

Discussion

datamodel interacts with the Splunk platform's existing data models, enabling efficient searches and analytics across large datasets by pre-defining schema and relationships.

```
| datamodel [data_model_name] [<dataset
name>] [<data model search mode>]
[strict_fields=<bool>] [allow_old_
summaries=<bool>] [summariesonly=<bool>]
```

Note: Search optimization can be further enhanced by accelerating your data models and interacting with your data through **tstats**!

Problem

You suspect unusual outbound traffic on port 1337 that may indicate potential exfiltration activity, but you don't know which indexes and sourcetypes contain relevant network data. You need to perform a quick analysis across all network logs to identify suspicious connections without manually specifying each data source.

Solution

Retrieve the count of events where destination port 1337 is observed in the `Network_Traffic` data model, grouped by source and destination IP addresses, and sort the results in descending order by count.

```
| tstats count summariesonly=true from
datamodel=Network_Traffic where All_
Traffic.dest_port=1337 by All_Traffic.
src_ip All_Traffic.dest_ip
| sort - count
```

Discussion

tstats is a generating command that works on indexed fields (host, source, sourcetype and _time) and existing data models.

```
| tstats <stats-func>
[summariesonly=<bool>] [from
datamodel=<data_model_name>] [where
<searchQuery>] [by <field-list>]
```

Problem

You want to view and graph the event volume of DNS events over the past seven days.

Solution

This query counts DNS events over the past seven days, grouping the results by host and time in one hour intervals. It then uses xyseries to pivot on the data, transforming the data so that each unique time value becomes a row, each host becomes a column, and then the count of events fills in the table, creating a time-series view of DNS activity by host.

```
| tstats count where index=network
sourcetype=stream:dns earliest=-7d by
_time, host, span=1h
| xyseries _time, host, count
```

Discussion

xyseries converts results into a tabular format that is suitable for graphing. This command is the inverse of the **untable** command.

```
...| xyseries [grouped=<bool>] <x-field>
<y-name-field> <y-data-field>...
[sep=<string>] [format=<string>]
```

```
...| untable <x-field> <y-name-field>
<y-data-field>
```

Problem

You want to create a baseline of visited domains and hunt traffic to *new* domains.

Solution

This query identifies newly observed domains by analyzing web traffic data from the Web data model and comparing it with previously seen domains stored in a lookup file, `previously_seen_domains.csv`. First, we must create this lookup with the search:

```
tag=web url=*
| eval list="mozilla"
| `ut_parse_extended(url,list)`
| stats earliest(_time) as earliest
latest(_time) as latest by ut_domain
| outputlookup previously_seen_domains.
csv
```

Following baseline creation, we compare the current web traffic domains to this lookup, identifying new domains that do not yet exist in the baseline or have only been observed recently:

```
| tstats count from datamodel=Web by
Web.url _time
| rename "Web.url" as "uri"
| eval list="mozilla"
| `ut_parse_extended(uri,list)`
| stats earliest(_time) as earliest
latest(_time) as latest by ut_domain
| inputlookup append=t previously_seen_
domains.csv
| stats min(earliest) as earliest
max(latest) as latest by ut_domain
| outputlookup previously_seen_domains.
csv
| eval isOutlier=if(earliest >= relative_
time(now(), "-1d@d"), 1, 0)
| convert ctime(earliest) ctime(latest)
| where isOutlier=1
```

****Special Ingredient:*** *the `ut_parse_extended` macro requires URL Toolbox, available on Splunkbase.*

Discussion

This hunting query is built to find new domains seen for the first time within the last 24 hours. Adjust the timing parameters as needed to reduce or expand the results set as needed for your hunt. During the first run of the query, many domains may be flagged as new since no historical data exists in the lookup file. Over time, as the baseline grows, the query will become more accurate and effective at detecting true outliers.

The **convert ctime()** function changes the timestamp to a non-numerical value, which is useful for display in a report or for readability in your events list.

Creating or updating fields on-the-fly

Problem

You need to evaluate and label network traffic as internal or external based on a condition.

Solution

This query looks at the firewall data set and uses eval to create a new field called `source_zone`. It uses the `cidr_match` function to check each event for whether the `src_ip` value falls into the internal IP range `10.0.0.0/8`. If the value falls into the range, it gives the value of "`internal`" to `source_zone`; otherwise, it assigns "`external`".

```
index=firewall
| eval source_zone=if(cidrmatch
("10.0.0.0/8", src_ip), "internal",
"external"
```

Discussion

eval calculates the value of an expression and assigns the result to a new field or existing field. It's versatile, supporting arithmetic, string, and conditional operations. If the field name that you specify matches an existing field name, the values in the existing field are replaced by the results of the eval expression.

```
...| eval <field1>=<expression1>[,
<field2>=<expression2>]
```

Problem

You are hunting for exceptionally-long command line entries.

Solution

This query examines the Windows Sysmon dataset for events containing the `CommandLine` field. It calculates the length of the `CommandLine` using the `len` function, storing the result in a new field called `lenCL`. It then filters for events where `lenCL` exceeds 1000 characters and displays the results in a table, sorting them in descending order by `lenCL`.

```
index=botsv1 sourcetype=XmlWinEventLo
g:Microsoft-Windows-Sysmon/Operational
CommandLine=*
| eval lenCL=len(CommandLine)
| where lenCL>1000
| table _time CommandLine ProcessId
ParentProcessId ParentCommandLine lenCL
| sort - lenCL
```

Discussion

len, used within eval, calculates the length of a string or number of characters in a field. It's often used to validate data lengths or for quality checks.

The `eval` command is incredibly versatile, and can compute bitwise functions, cryptographic functions, or evaluate other characteristics of text — be sure to check the docs and quick reference guide for more uses!

2. Sorting and stacking

Identifying high-volume activity

Problem

You want to identify the top talkers to external hosts.

Solution

```
index=proxy src=192.168.225.* NOT
(dest=192.168.* OR dest=10.* OR
dest=8.8.4.4 OR dest=8.8.8.8 OR
dest=224.*)
| stats sum(bytes_in) as total_bytes_in
sum(bytes_out) as total_bytes_out by
src dest
| table src dest total_bytes_in total_
bytes_out
| sort – total_bytes_out
```

Discussion

stats calculates aggregate statistics, such as count, average, and sum, over the results set.

```
...| stats <stats-function>(<wc-field>)
[as <wc-field>] [by <field-list>]
```

Function type table

Type of function	Supported functions and syntax			

Aggregate functions	avg()
	count()
	distinct_count()
	estdc()
	estdc_error()
		exactperc<num>()
		max()
		medium()
		min()
		mode()
			perc<num>()
			range()
			stdev()
			stdevp()
				sum()
				sumsq()
				upperperc<num>()
				var()
				varp()

Event order functions	first()	last()		
Multivalue stats and chart functions	list()	values()		

Time functions	earliest()
	earliest_time()
		latest()
		latest_time()
			rate()	

Problem

You want to identify the most common source address of network traffic generation.

Solution

This query identifies the source IP addresses with the top count in the network index.

```
index=network
| top src_ip
```

Discussion

top returns the most frequent values in a given field. It is often used to quickly identify the most common occurrences in data sets.

```
...| top (<field>|<field-list>) [by
<field>] [countfield=<string>]
[limit=<int>] [showperc=<bool>]
```

Isolating rare events and values

Problem

You need to hunt for rare user accounts logging into critical systems.

Solution

This query identifies users with the fewest occurrences in the authentication dataset.

```
index=authentication
| rare user
```

Discussion

rare is the opposite of top; it finds the least common values in a field, highlighting anomalies or rare events in a data set. `top` and `rare` can be useful while **baselining** or developing searches to **test hypotheses**.

```
...| rare (<field>|<field-list>)
```

Sorting results

Problem

You need to organize results for a numerical or categorical field. For example, organizing results for the `_time` field to time-order investigation results.

Solution

This query retrieves authentication events and sorts them from the most recent to the oldest.

```
index=authentication
| sort - _time
```

Discussion

sort arranges search results in ascending or descending order based on specified fields. It helps in organizing data for better analysis and readability. Use a minus sign (-) for descending order and a plus sign (+) for ascending order.

```
...| sort (-|+) <field> [limit=<int> |
<int>
```

Note: `sort` will return 10,000 events if no `limit` is specified. To return all results, use: "`sort limit=0`".

Calculating search-time statistics

Problem

You need to detect which user accounts are generating the most failed login attempts across multiple systems *and still retain the original event details* for investigation.

Solution

This query allows you to see both the count of failures per user and the individual event data (e.g., the specific hosts, timestamps, etc.) for each failed login attempt.

```
sourcetype=authentication
status="failure"
| eventstats count AS total_failed_
attempts by user
| table _time user host status total_
failed_attempts
| sort - total_failed_attempts
```

Discussion

If you were to use `stats` instead of `eventstats`, the query would aggregate the events, losing the individual event details (such as timestamps, host information, etc.). Instead, "`| eventstats count AS total_failed_attempts by user`" counts the number of failed login attempts per user and stores the result in `total_failed_attempts`. This command preserves individual event details while adding the summary field.

`eventstats` generates summary statistics from fields in your events and saves those statistics in a new field.

```
...| eventstats [allnum=<bool>] <stats-
agg-term> … [<by-clause>]
```

Problem

You want to identify any source IPs that have triggered multiple multifactor authentication (MFA) failures in a short time period in your authentication data source.

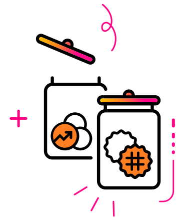
Solution

```
index=authentication eventType=mfa
action=failed
| streamstats count by src_ip user
| where count > 5
| stats values(user) by src_ip
```

Discussion

`streamstats` builds upon the basics of the stats command but it provides a way for statistics to be generated as each event is seen.

```
streamstats [reset_on_change=<bool>]
[reset_before="("<eval-expression>")"]
[reset_after="("<eval-expression>")"]
[current=<bool>] [window=<int>] [time_
window=<span-length>] [global=<bool>]
[allnum=<bool>] <stats-agg-term>...
[<by-clause>]
```



3. Grouping

Grouping by field values

Problem

You need a list of unique `dest_ip` by user account.

Solution

```
index=network user="jdoe"
| stats values(dest_ip) by user
```

Reviewing field values by group

Problem

You need to establish context in Microsoft Azure AD Events.

Solution

```
index=azure sourcetype="ms:aad:audit"
activity=*user*
| stats values(activityOperationType)
values(targets{}.userPrincipalName)
count by actor.userPrincipalName
```

Discussion

`values` returns a list of unique values for a given field. It is often used with aggregation commands to get distinct values from grouped data.

Counting by group

Problem

You want to search for a variety of discovery-related commands, and total via distinct count for potential enumeration activity.

Solution

```
index=authentication
| eventstats dc(enumerationActions) AS
actionsCount by userName
| sort - count
```

Discussion

`dc` calculates the distinct count of values in a field. It's useful for identifying the number of unique items in a dataset.

```
...| stats distinct_count(value) | stats
dc(<value>)
```

Merging groups by a common field

Problem

You need to correlate different types of events that share a common identifier, such as a user ID, hostname, or IP address. For instance, you may be hunting a hypothesis where a user accesses a sensitive file within 10 minutes of logging in.

Solution

This query uses the `join` command to combine two different data sources, authentication events and file access events, then correlates logins with sensitive file access. The `join` ensures only users present in both datasets are included, allowing the calculation of the time difference between login and file access. It then filters for cases where files were accessed within 600 seconds of the login, helping identify potentially suspicious behavior.

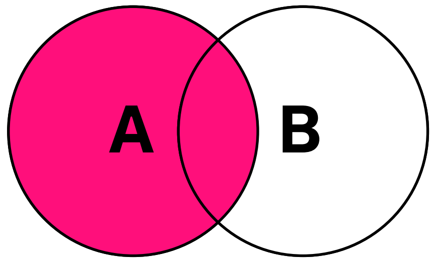
```
index=authentication sourcetype=login
| table user, _time, src_ip
| rename _time as login_time
| join user [
    search index=security
    sourcetype=file_access sensitive_
    file="*"
    | table user, _time, file_path
    | rename _time as access_time
]
| eval time_diff = access_time - login_
time
| where time_diff >= 0 AND time_diff <=
600
| table user, src_ip, login_time,
access_time, file_path
```

Discussion

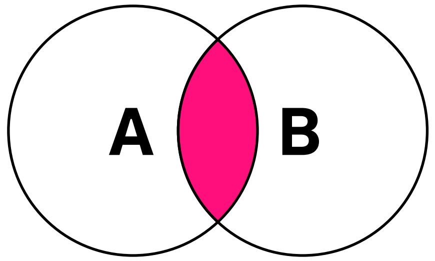
join combines results from two different searches based on a common field. The results of a main search are considered the “left-side” dataset, while the subsearch represents the “right-side” dataset. By default, `join` performs an inner `join`, returning only events from the main search that have matches in the subsearch. A left (or outer) `join` includes all events from the main search, along with only the matching values from the subsearch.

```
join [<join-options>...] [<field-
list>] | [left=<left-alias>]
[right=<right-alias>] where <left-
alias>.<field>=<right-alias>.<field>
[<left-alias>.<field>=<right-
alias>.<field>]...<dataset-type>:<dataset-
name> | <subsearch>
```

Left Join



Inner Join



Problem

You want to track login attempts that happen close together across different systems by the same user, which could indicate lateral movement.

Solution

This query uses the `transaction` command to group events for each user, focusing on sequences of failed login attempts followed by a successful one within a 10-second window, with no more than 5 seconds between events. It then filters for sequences with more than 3 events and displays the results in a table.

The `transaction` command is ideal here because it allows setting constraints like `maxspan` and `maxpause` to capture only short bursts of login activity.

```
index=authentication sourcetype=login
(action="failure" OR action="success")
| transaction user
maxspan=10s maxpause=5s
startswith=(action="failure")
endswith=(action="success")
| search eventcount > 3
| table _time user host
eventcount duration
```

Discussion

transaction allows you to group together related events that share a common field or fields and occur within a specified time window. It is especially useful when you want to track a sequence of events that are related but span multiple logs or event types.

```
| transaction [<field-list>]
[name=<transaction-name>] [<txn_
definition-options>...] [<memcontrol-
options>...] [<rendering-options>...]
```

4. Forecasting and anomaly detection

Comparing anomaly calculation methods

Standard Deviation Method

- **Definition:** A data point is considered an outlier if it lies more than a certain number of standard deviations away from the mean.
- **Threshold:** Common thresholds are 2, 2.5, or 3 standard deviations from the mean.
- **Example:** In a normal distribution, approximately 95% of the data lies within 2 standard deviations of the mean. Data points beyond this range may be considered outliers.
- **SPL:** Calculate a standard deviation from a counted metric, and set bounds at ±2 times the standard deviation:

```
| stats stdev(count) as stdev
| eval lowerBound=(avg-stdev*2)
| eval upperBound=(avg+stdev*2)
| eval isOutlier=if(count < lowerBound
OR count > upperBound, "Yes", "No")
```

Interquartile Range (IQR) Method

- **Definition:** An outlier is defined as a data point that lies outside of a specified range determined by the IQR.
- **Threshold:** Data points are typically considered outliers if they fall below Q1 – 1.5 × IQR or above Q3 + 1.5 × IQR. Here, Q1 and Q3 represent the first and third quartiles, respectively, and IQR stands for the interquartile range.
- **Example:** The IQR is the range between the first and third quartiles (25th and 75th percentiles). Outliers are often defined as data points falling more than 1.5 times the IQR below the first quartile or above the third quartile.
- **SPL:** This search calculates the first quartile (Q1) and third quartile (Q3), computes the interquartile range (IQR), and then filters outliers that fall outside the bounds of 1.5 times the IQR below Q1 or above Q3:

```
| stats perc25(value) AS Q1,
perc75(value) AS Q3
| eval IQR = Q3 - Q1
| eval lower_bound = Q1 - 1.5 * IQR
| eval upper_bound = Q3 + 1.5 * IQR
| where value < lower_bound OR value
> upper_bound
```

Z-Score Method

- **Definition:** The Z-score measures the number of standard deviations a data point is from the mean.
- **Threshold:** A common threshold for outliers is a Z-score greater than 3 or less than -3.
- **Example:** If a data point has a Z-score of 4, it is 4 standard deviations away from the mean, making it a potential outlier.
- **SPL:** This search calculates the mean and standard deviation of the `value` field and then computes the Z-score for each data point. Outliers are identified where the absolute value of the Z-score exceeds 3:

```
| stats avg(value) AS mean, stdev(value)
AS stdev
| eval z_score = (value - mean) / stdev
| where abs(z_score) > 3
```

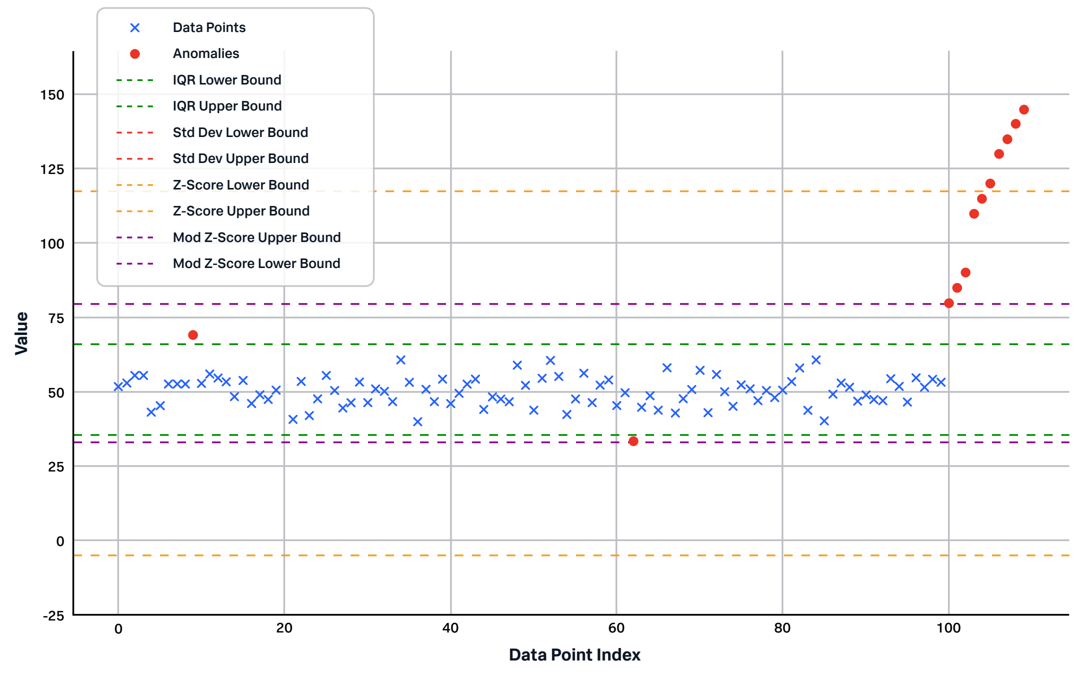
Modified Z-Score

- **Definition:** Similar to the Z-score, but uses the median and median absolute deviation (MAD) instead of the mean and standard deviation.
- **Threshold:** A common threshold for outliers using the modified Z-score is 3.5 or higher.
- **Example:** If a data point has a modified Z-score of 4.2, it is far from the median and is likely an outlier based on the threshold of 3.5 or higher.
- **SPL:** In this search, the eventstats command calculates the median and MAD of the value field. The MAD is adjusted using a constant (1.4826) to make it comparable to standard deviation. The modified Z-score is then calculated, and outliers are identified where the absolute modified Z-score exceeds 3.5:

```
| eventstats median(value) AS median_
value, stdev(value) AS mad
| eval mad = 1.4826 * mad
| eval modified_z_score = 0.6745 *
(value - median_value) / mad
| where abs(modified_z_score) > 3.5
```

Here we can see a comparison of the different anomaly detection methods. Notice how *the classification of an outlier changes based on the method in use!*

Anomaly Detection Methods



Using built-in anomaly detection

Problem

You want to hunt for abnormal login activity by calculating outliers in a user's logon times.

Solution

Use built-in probability modeling to find rare login occurrences. This query searches Windows Security logs for EventCode=4624 (successful logon events) for the user "buttercup" using a 1 hour time span. It then applies the anomalydetection command to identify unusual deviations in login activity.

```
index=authentication sourcetype=
"WinEventLog:Security" EventCode=4624
Account_Name=buttercup
| timechart span=1h count
| anomalydetection
```

Alternatively, we can manually compute an outlier via the IQR method, so we have more control over the threshold, e.g.: finding outliers in Windows Logon Events:

```
index=authentication
sourcetype="WinEventLog:Security"
EventCode=4624
| eventstats avg("_time") as avg
stdev("_time") as stdev
| eval lowerBound=(avg-stdev*exact(2)),
upperBound=(avg+stdev*exact(2))
| eval isOutlier=if('_time' < lowerBound
OR '_time' > upperBound, 1, 0)
| table _time, isOutlier, body
```

Discussion

anomalydetection identifies anomalies in data based on built-in statistical methods. This method is probability-based, and anomalies are flagged when they have an exceptionally low probability of occurring within the set of results. This command creates several new fields based on the method and action parameters used. The method options include histogram (default), Z-score, and IQR.

- For categorical fields, the frequency of a value X is the number of times X occurs divided by the total number of events.
- For numerical fields, we first build a histogram for all the values, then compute the frequency of a value X as the size of the bin that contains X divided by the number of events.

timechart applies a statistical aggregation of the variable you specify after the BY clause, using your timestamp as the X-axis on the chart. This is a useful transformation to look at changes in a variable with respect to time.

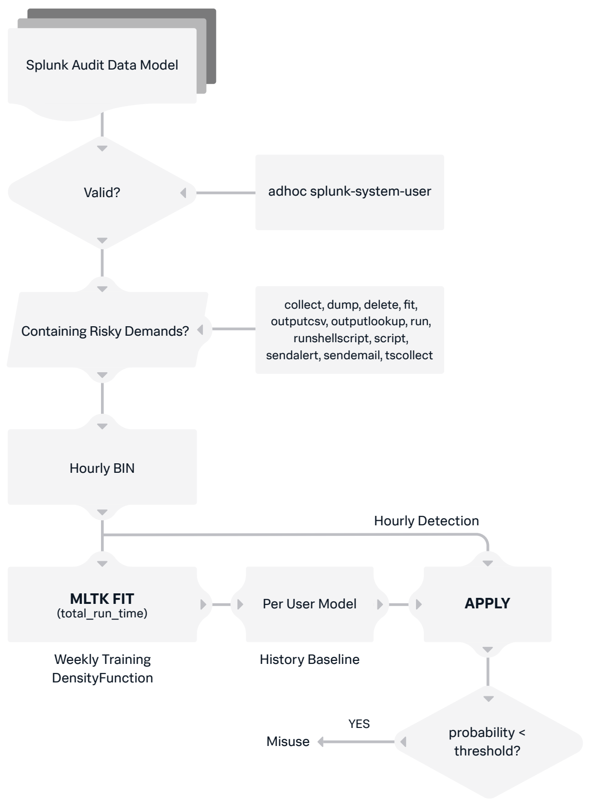
Model-assisted methods

The Splunk Machine Learning Toolkit (MLTK), and the Splunk App for Data Science and Deep Learning (DSDL) offer advanced data science methods and algorithms for use on your Splunk data. MLTK uses the `fit` and `apply` commands to enable model training and inference, respectively. The Splunk App for DSDL also provides access to these algorithms through the same SPL, but within the added context of a Jupyter Lab environment for added extensibility to manipulate your data or import and connect with resources external to Splunk.

Forecasting and anomaly detection algorithms detect anomalies and outliers in numerical or categorical fields, or within the learned characteristics of a time-series.

- **DensityFunction** provides a consistent and streamlined workflow to create and store density functions and utilize them for anomaly detection. The algorithm converts time series data into a symbolic representation to detect anomalies by comparing against normal patterns. DensityFunction allows for grouping of the data using the by clause, where for each group a separate density function is fitted and stored. This algorithm supports incremental fit.
  - Detection of Risky Command Exploit uses the MLTK DensityFunction to analyze command sequences and calculate their likelihood based on learned distributions. Hunters can focus on events that fall outside the expected density as suspicious, potentially malicious occurrences.
- **OneClassSVM** is trained on normal data to identify anomalies by determining whether new data points fall outside the learned distribution. OneClassSVM is an unsupervised outlier detection method.

Detection Flow Chart



- **Autoregressive Integrated Moving Average (ARIMA)** uses the StatsModels ARIMA algorithm to fit a model on a time series for better understanding and/or forecasting its future values. An ARIMA model can consist of autoregressive terms, moving average terms, and differencing operations. The autoregressive terms express the dependency of the current value of time series to its previous ones.
- **StateSpaceForecast** is a forecasting algorithm for time series data in MLTK. It is based on Kalman filters. The algorithm supports incremental fit.
  - Advantages of StateSpaceForecast over ARIMA include:
    - Persists models created using the fit command that can then be used with apply.
    - A specialdays field allows you to account for the effects of a specified list of special days.
    - It is automatic in that you do not need to choose parameters or mode.
    - Supports multivariate forecasting.

Fitting an anomaly detection model

Problem

You want to fit an anomaly detection model to detect spikes in SMB traffic.

Solution

First, fit a model to the normal pattern of SMB traffic. The **fit** command generates a statistical model based on historical traffic patterns. This format analyzes SMB traffic using port 445, or port 139, and establishes patterns based on 10 minute intervals, enriching the data by finally including the hour of the day and day of the week:

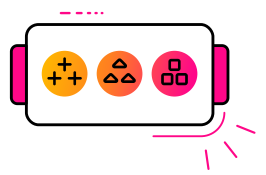
```
| tstats summariesonly=false allow_
old_summaries=true count from
datamodel=Network_Traffic where All_
Traffic.dest_port=139 OR All_Traffic.
dest_port=445 OR All_Traffic.app=smb by
_time span=10m, All_Traffic.src
| eval HourOfDay=strftime(_time, "%H")
| eval DayOfWeek=strftime(_time, "%A")
| rename "All_Traffic.*" as *
| fit DensityFunction count by
"HourOfDay,DayOfWeek" into smb_pdfmodel
```

The **apply** command uses this model to detect outliers from the learned patterns in 1-hour intervals. Results are sorted by the highest count of outliers.

```
| tstats summariesonly=false allow_
old_summaries=true count values(All_
Traffic.dest_ip) as dest values(All_
Traffic.dest_port) as port from
datamodel=Network_Traffic where All_
Traffic.dest_port=139 OR All_Traffic.
dest_port=445 OR All_Traffic.app=smb by
_time span=1h, All_Traffic.src
| eval HourOfDay=strftime(_time, "%H")
| eval DayOfWeek=strftime(_time, "%A")
| rename "All_Traffic.*" as *
| apply smb_pdfmodel threshold=0.001
| rename "IsOutlier(count)" as isOutlier
| search isOutlier > 0
| sort - count
| table _time src dest port count
```

Discussion

The use of the `DensityFunction` and time-based features allows for a model that adapts to the regular traffic patterns specific to the environment. Since SMB is commonly used for file sharing and remote access, abnormal spikes in SMB traffic could indicate malicious activity, such as lateral movement across the network or scanning for vulnerable SMB services, or exfiltration activity. The threshold can be adjusted to increase or decrease sensitivity to outliers.



5. Clustering

Grouping via categorical fields

Problem

You need to group similar user-agent strings, which are non-numeric and categorical, across network traffic to detect patterns of suspicious activity.

Solution

Cluster similar `user_agent` strings from firewall events based on a 90% similarity threshold (`t=0.9`). This value means user-agents need to be 90% similar to be grouped in the same cluster. Then it provides a summary of the source and destination IPs, user-agent, assigned cluster label (`cluster_label`), and the event count for each cluster (`cluster_count`).

```
index=firewall user_agent=*
| fields src_ip dest_ip user_agent
| cluster t=0.9 showcount=true
field=user_agent labelonly=true
| table src_ip dest_ip user_agent
cluster_label cluster_count
```

Discussion

cluster groups events into clusters based on field values, helping identify patterns and similarities in data. It is useful for categorizing data into meaningful groups, for example, cohorts of similar users, types of web traffic, or browser user-agents.

```
| cluster t=<num> | delims=<string> |
showcount=<bool> | countfield=<field>
| labelfield=<field> | field=<field> |
labelonly=<bool> | match=(termlist |
termset | ngramset)
```

fields keeps or removes fields from search results based on the field list criteria. By default, the internal fields `_raw` and `_time` are included in output. Limiting the number of fields with fields speeds up searches by reducing the data the Splunk platform processes and returns, especially in large datasets.

```
...| fields [+|-] <wc-field-list>
```

Problem

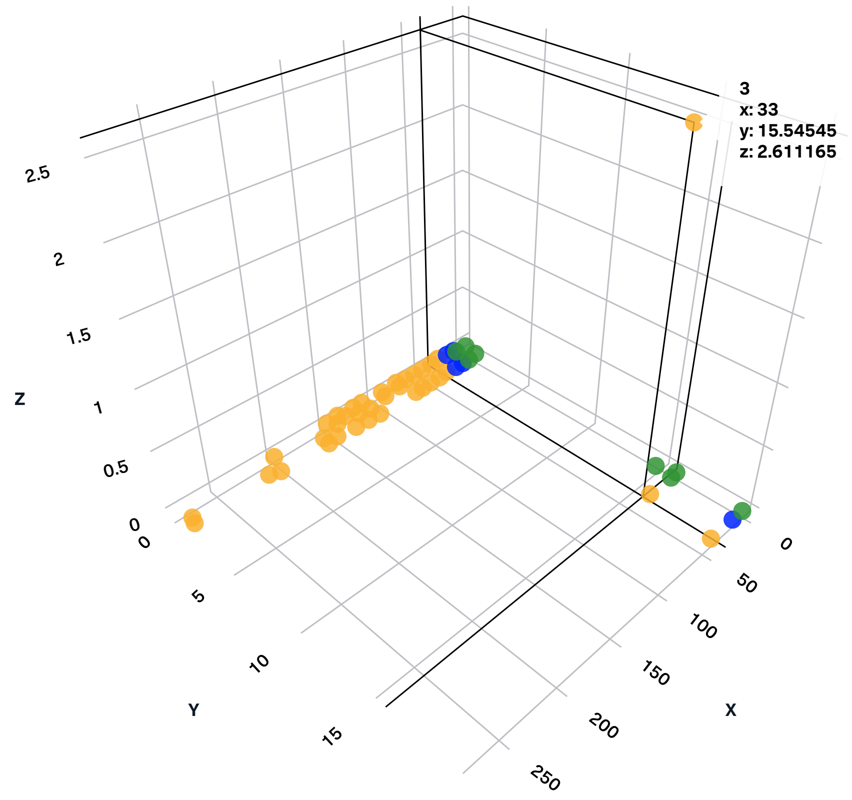
You need to analyze and group source IP addresses by their numeric risk scores and event counts to identify clusters of potentially high-risk sources.

Solution

This query analyzes the risk index by calculating the event count, average risk score, and standard deviation (quantifies how much the `risk_score` values deviate from the mean (`avg_risk_score`)) for each source IP. Using these metrics, it applies the K-Means algorithm to group the data into 3 clusters based on event frequency and risk score patterns. These clusters help to highlight outliers or sources with abnormal risk behaviors.

```
index=risk
| stats count as event_count, avg(risk_
score) as avg_risk_score, stdev(risk_
score) as stdev_risk_score by src_ip
| kmeans k=3 event_count avg_risk_score
stdev_risk_score
| table src_ip, event_count, avg_risk_
score, stdev_risk_score, CLUSTERNUM
| rename event_count as x, avg_risk_
score as y, stdev_risk_score as z,
CLUSTERNUM as clusterId
```

Graphed Clusters via 3-D Scatterplot



Discussion

kmeans partitions numeric events into k clusters based on their mean values. Each event belongs to the cluster with the nearest mean value. Clustering is performed on specific fields, or on all numeric fields by default. It is useful for grouping similar events, helping to identify anomalies, hidden patterns, and outliers indicating malicious activity.

```
...| kmeans [kmeans-options...]
[field-list]
```

Pivoting a statistics table

Problem

You want to investigate the most common and least common combinations of processes launching from each application in your Windows Environment.

Solution

Calculate the count for each combination of process and new process names, and convert it to a pivot table. This allows comparison of the most common and least common combinations.

```
index=windows
sourcetype="WinEventLog:Security"
EventCode=4688
| stats count by Creator_Process_Name,
New_Process_Name
| xyseries Creator_Process_Name, New_
Process_Name, count
```

This query takes Windows Security logs with EventCode=4688 (new process creation events). The `xyseries` command transforms the relationship between `Creator_Process_Name` and `New_Process_Name` into a matrix, where rows represent parent processes and columns represent the processes they create. This structure simplifies the visualization and analysis of process creation patterns, making it easier to spot unusual behavior.

Discussion

xyseries converts search results into a table format with X and Y axes, useful for creating pivot tables or time series data representations.

- The x-field (first field listed) is the column; in other words, the x-axis field.
- The y-field (second field listed) will become the new column values, otherwise referred to as new field names.
- The y-data-field (third field listed) will be the data values in the table for the new fields you created.

```
xyseries [grouped=<bool>] <x-field>
<y-name-field> <y-data-field>...
[sep=<string>] [format=<string>]
```

Model-assisted methods

The Splunk Machine Learning Toolkit (MLTK) and the Splunk App for Data Science and Deep Learning (DSDL) offer advanced data science methods and algorithms for use on your Splunk data. MLTK uses the `fit` and `apply` commands to enable model training and inference, respectively. The Splunk App for DSDL also provides access to these algorithms through the same SPL, but within the added context of a Jupyter Lab environment for added extensibility to manipulate your data or import and connect with resources external to Splunk.

Clustering Algorithms implement different methods to assign groups to data points:

- **K-Means** partitions data into K distinct clusters based on feature similarity. It is one of the most commonly used clustering techniques. The cluster for each event is set in a new field named `cluster`. Use the K-means algorithm when you have unlabeled data and have at least approximate knowledge of the total number of groups into which the data can be divided. In a previous section, we used the `kmeans` command as another method to apply this algorithm in the Splunk platform.
- **Density-Based Spatial Clustering of Applications with Noise (DBSCAN)** groups data points that are closely packed together while marking outliers as noise. This is useful for identifying clusters of arbitrary shapes. DBSCAN is distinct from K-Means in that it clusters results based on local density, and uncovers a variable number of clusters, whereas K-Means finds a precise number of clusters. For example, k=5 finds 5 clusters.

6. Exploratory data analysis and visualization

Transforming data

Problem

Your visualization is not rendering correctly because you need your row values to be your column values.

Solution

Use transpose to rotate a table 90 degrees, turning rows into column headers, and column values into row items.

```
index=_internal component=Metrics
group=thruput
| stats sum(*kbps) as *kbps by host
| transpose 0 column_name=metric
header_field=host
```

By adding transpose to this search, the visualization groups the metrics together instead of the hosts, which allows for host over host comparison of the same metric.

Discussion

`transpose` returns the specified number of rows (search results) as columns (list of field values), such that each search row becomes a column. This command has no required arguments and a few optional arguments: column_name, header_field, include_empty, and int:

- `column_name` defines the name of the first column in the "new" table, defaults to "column"
- `header_field` defaults to "row 1," "row 2," etc., and is the field used to create the new columns' headers
- `include_empty` specifies if empty values will be included or not, defaults to true
- `int` specifies how many column headers to create, defaults to 5. With the int argument, 0 is unlimited and you do not need to type int=10, just the number 10 will suffice, and it will bring the first 10 rows in the table.

Exploring new data sources

Problem

You need to identify fields and sample values in the endpoint data source when you are scoping your hunt.

Solution

```
index=endpoint
| fieldsummary maxvals=2
```

Discussion

`fieldsummary` calculates summary statistics for all fields or a subset of the fields in your events. The summary information is displayed as a results table. Use the `maxvals` argument to specify the number of values returned.

```
| fieldsummary [maxvals=<unsigned_int>]
[<wc-field-list>]
```

Problem

You'd like to quickly list available sourcetypes and their fields.

Solution

```
| metadata type=sourcetype
```

Problem

You need to find all hosts writing to a specific index.

Solution

```
| metadata type=hosts index=<index_
name>
```

Problem

You want to find hosts that haven't reported in the last 24 hours.

Solution

```
| metadata type=hosts
| eval "Last Seen" = now() - recentTime
| where "Last Seen" > "86400"
| rename totalCount as count firstTime
as "First Event" lastTime as "Last
Event" recentTime as "Last Update"
| fieldformat "First Event" =
strftime('First Event', "%c")
| fieldformat "Last Event" =
strftime('Last Event', "%c")
| fieldformat "Last Update" =
strftime('Last Update', "%c")
| eval "Minutes Behind" = round('Last
Seen'/60, 2)
| eval "Hours Behind" = round('Last
Seen'/3600, 2)
| table host, "First Event", "Last
Event", "Last Update", "Hours Behind",
"Minutes Behind"
| sort -"Minutes Behind"
```

Discussion

`metadata` can be searched and returned with values that include first time, last time and count for a particular value. This is a generating command, which means it is the first command in a search. This command can be particularly useful in baselining hunts.

```
| metadata type=<metadata-type>
[<index-specifier>]... [splunk_
server=<wc-string>] [splunk_server_
group=<wc-string>]...<datatype>
```

Uncovering relationships between variables

Problem

You want to develop and evaluate the accuracy of a predictive model that forecasts a trend in your data. Specifically, you need to compare the actual values of 'future logons' with the predicted values to assess how well the model performs.

Solution

For a more detailed look at future prediction methods, check out the **Forecasting and Anomaly Detection** section. Assuming we have already trained a forecasting model, we can evaluate the fit and accuracy of our output (`logon_data.csv`), via:

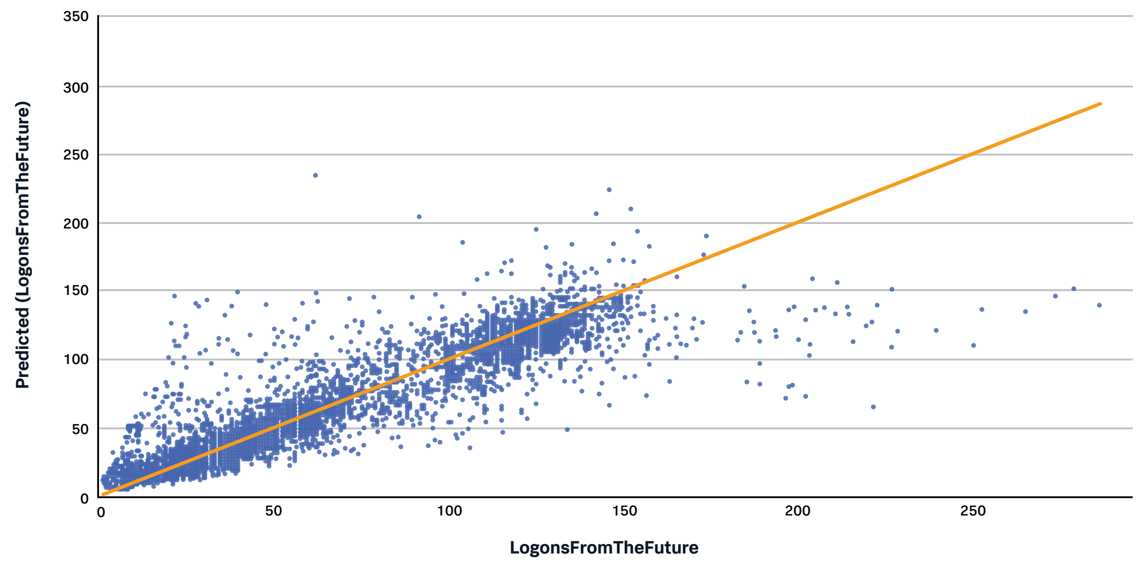
```
| inputlookup logon_data.csv
| eval prediction_error =
abs(LogonsFromTheFuture - predicted_
LogonsFromTheFuture)
| trendline linear(LogonsFromTheFuture)
as trendline
| table LogonsFromTheFuture, predicted_
LogonsFromTheFuture, trendline,
prediction_error
```

The **scatter plot** (“Actual vs. Predicted Scatter Chart” pictured below) is a critical aide to help you visualize and interpret the relationship between two numerical values.

Discussion

Scatter plots are a graphical technique used to analyze the relationship of two numeric variables. You can fit a linear regression to demonstrate the relationship making it easier to identify an outlier, or evaluate the performance of your predictions.

Actual vs. Predicted Scatter Chart



Problem

You want to analyze web traffic logs, and create a box plot to visualize the distribution of URL lengths across the dataset to detect outliers.

Solution

```
index=network sourcetype=stream:http
| eval url_length = len(url)
| fields url_length
| `boxplot`
```

***Special Ingredients:** The `boxplot` macro and the Box Plot visualization are bundled with the Splunk MLTK. You must be in the app to use this macro.*

Discussion

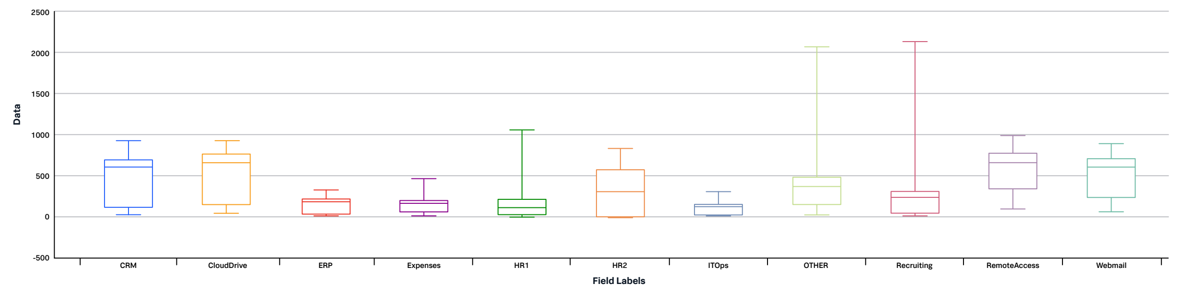
Box plots are used to show the minimum, lower quartile, median, upper quartile, and maximum of each field, which can be particularly useful in comparing values across categories, user accounts, or hosts. A box plot or box and whisker plot graphically depicts groups of numerical data through their quartiles, with lines (whiskers) extending vertically to indicate the upper and lower quartiles. Box plots calculate and visualize key summary statistics for baselining hunts.

A box plot is developed from five key statistics (also known as the five-number summary):

- Minimum value — the smallest value in the data set
- Second quartile — the value below which the lower 25% of the data are contained
- Median value — the middle number in a range of numbers
- Third quartile — the value above which the upper 25% of the data are contained
- Maximum value — the largest value in the data set

When you have no idea what normal looks like for a set of numeric data (e.g., URL length), a box plot can be useful to visualize expected activity versus outliers. Everything within the box (from the second to third quartile, or 25-75% of the data) is considered normal activity, while the whiskers represent outliers.

Box Plot Chart



Visualizing distributions

Problem

You want to visualize the distribution of network traffic based on protocol.

Solution

```
index=network
| chart count by protocol
```

Discussion

chart returns your results in a table format. The results can then be used to display the data as a chart, such as a column, line, area, or pie chart.

```
...| chart <stats-func>(<wc-field>)
over <row-split> [by <column-split>]
[span=<int><timescale>][limit=<int>]
[useother=<bool>] [usenull=<bool>]
```

Problem

You need to group and count the distribution of user-agent lengths in your proxy logs.

Solution

Understand the shape and count of categorical variables in your data.

```
index=proxy user_agent=*
| bin "http_user_agent_length" bins=100
| stats count by "http_user_agent_
length"
```

Discussion

bin puts continuous numerical values into discrete sets, or bins, by adjusting the value of a field so that all of the items in a particular set have the same value. `Bucket` is an alias of the `bin` command.

```
| bin [<bin-options>...] <field>
[AS <newfield>]
```

Alternatively, the same SPL query can be shortened using the `histogram` macro in the MLTK app: `` `histogram(<field, bins>)` ``

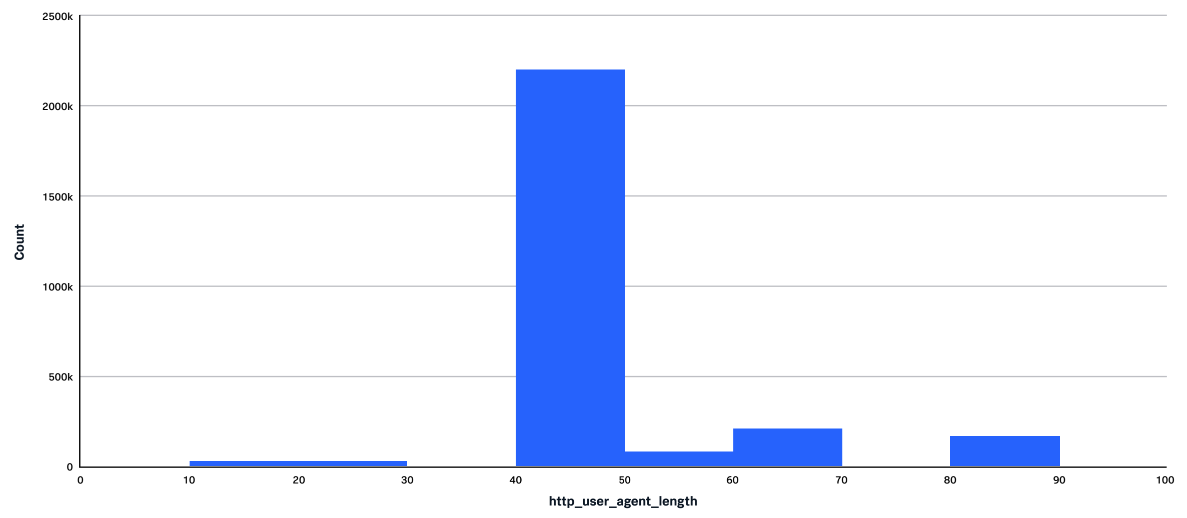
```
index=proxy user_agent=*
| `histogram(http_user_agent_length,
100)`
```

This expands to:

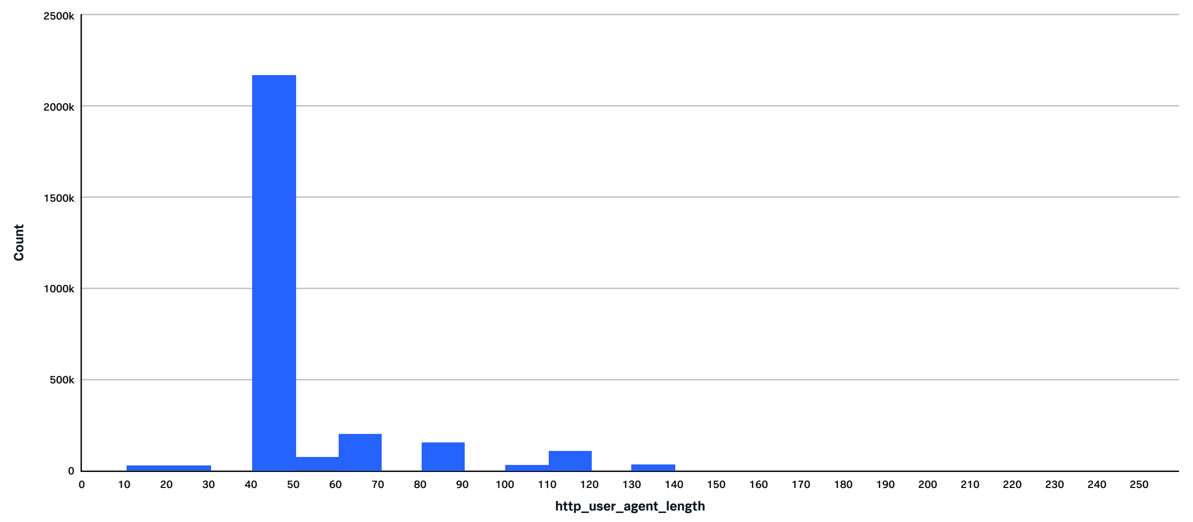
```
index=proxy user_agent=*
| bin "http_user_agent_length" bins=100
| stats count by "http_user_agent_
length"
| makecontinuous "http_user_agent_
length"
| fillnull count
```

The additional commands provided by the macro (e.g. **makecontinuous**) ensure there are no gaps in the range and handle missing counts for a better visualization.

Histogram Chart: `chart`



Histogram Chart: `bin`



Working with _time

Problem

Count failed login attempts by users over the past day.

Solution

```
index=authentication status="failure"
| timechart span=1d count by user
```

Discussion

`timechart` creates time series charts, aggregating data over specified time intervals. It is particularly useful for identifying trends, or visualizing changes in quantity, like event volume.

```
...| timechart <stats-func>(<field>) by
<split-by-field>
[span=<int><timescale>] [limit=<int>]
```

Problem

You want to create a dashboard panel search with a visual indicator of spiking login attempts to the AWS Console.

Solution

```
index=aws sourcetype=aws:cloudtrail
eventName=ConsoleLogin
errorMessage="Failed Authentication"
eventSource=signin.amazonaws.com
| bucket _time span=1d AS bucket
| eval
ateStamp=strftime(bucket,"%m/%d/%y")
| stats count sparkline by dateStamp,
eventName, userIdentity.userName
errorMessage
| sort -count
| search count >15
```

Discussion

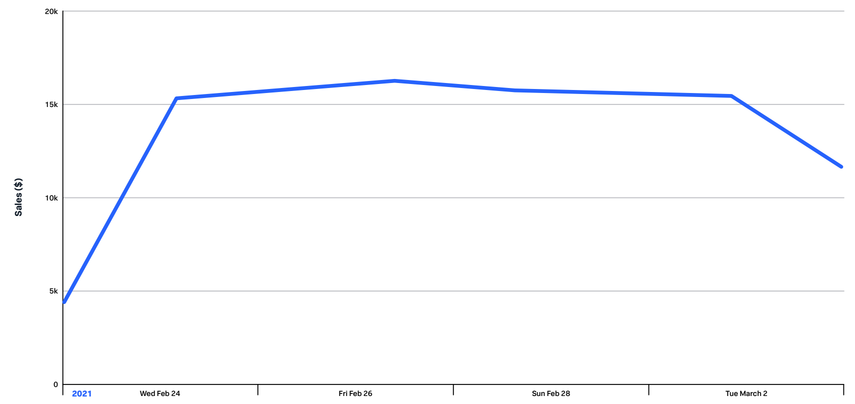
`sparkline` creates small inline charts that show trends in data, typically used within tables for quick visual insights.

```
| stats [partitions=<num>]
[allnum=<bool>] [delim=<string>] (
<stats-agg-term>... | <sparkline-
agg-term>... ) [<by-clause>] [<dedup_
splitvals>]
```

Time Chart

```
index=_internal
| timechart span=1d count
```

_time	count
2021-02-24	4250
2021-02-25	15256
2021-02-26	15871
2021-02-27	16194
2021-02-28	15752
2021-03-01	15535



7. Combined methods

Advanced recipes

Problem

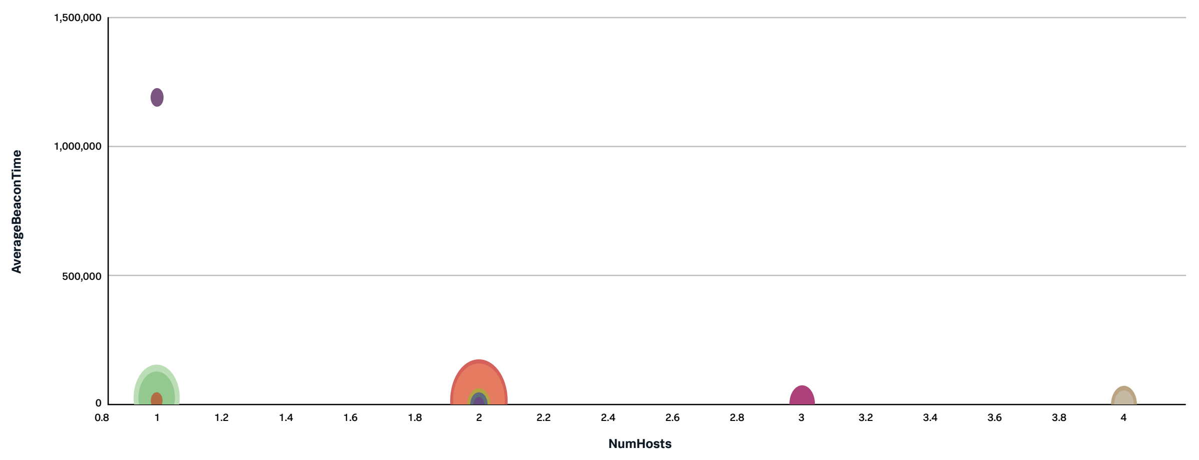
You suspect Command and Control (C2) beaconing activity would be visible through your DNS data source.

Solution

Searching for clients that show a low variance in time between connections may indicate hosts are contacting C2 infrastructure on a predetermined time slot, e.g., every thirty seconds or every five minutes.

```
tag=dns message_type="QUERY"
| fields _time, query
| streamstats current=f last(_time) as
last_time by query
| eval gap=last_time - _time
| stats count avg(gap) AS
AverageBeaconTime var(gap) AS
VarianceBeaconTime BY query
| eval AverageBeaconTime=round
(AverageBeaconTime,3),
VarianceBeaconTime=round
(VarianceBeaconTime,3)
| sort -count
| where VarianceBeaconTime < 60 AND
count > 2 AND AverageBeaconTime>1.000
| table query VarianceBeaconTime count
AverageBeaconTime
```

Bubble Chart



Discussion

This solution aims to detect potential Command and Control (C2) beaconing activity by identifying DNS queries that exhibit a consistent time pattern. Using the **`streamstats`** command, it calculates the time gap between consecutive DNS queries for each unique query value. The **`stats`** command is then applied to compute the average time (`AverageBeaconTime`) and the variance in time (`VarianceBeaconTime`) between these queries.

By filtering results to include only those with a low variance (`VarianceBeaconTime < 60`) and a sufficient number of occurrences (`count > 2`), this query helps pinpoint queries that are made at regular intervals — potential indicators of automated C2 communication.

Problem

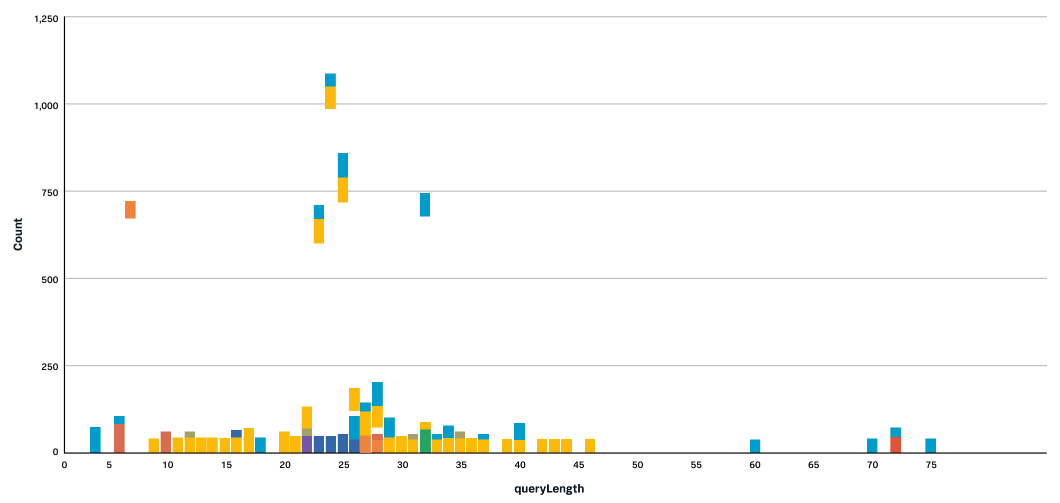
Your hypothesis suggests that DNS-based exfiltration would stand out in a graph comparing high packet size vs. high event volume.

Solution

Unusually large DNS queries may indicate that data is being encoded and sent out through DNS requests. The **`mvexpand`** command expands multi-value `query` fields, then calculates the length of each query using `len(query)`. The `stats` command aggregates the count of queries by `queryLength` and source (`src`). Sorting by `queryLength` and `count` allows the detection of sources that are generating a high volume of long DNS queries, which could be indicative of exfiltration attempts.

```
tag=dns message_type="QUERY"
      | mvexpand query
      | eval queryLength=len(query)
      | stats last(query) as sample
count by queryLength, src
      | sort -queryLength, count
      | table src queryLength count
      | head 1000
```

Scatter Chart



Discussion

By focusing on the top 1000 results with the head command, this approach surfaces the most suspicious DNS activity. Analysts can then use this table to identify and investigate the source systems (`src`) that might be involved in exfiltrating data through DNS, especially if their queries deviate significantly in length or frequency compared to normal behavior.

Problem

You want to visualize the geographic distribution of source IPs using AWS access keys to identify anomalies.

Solution

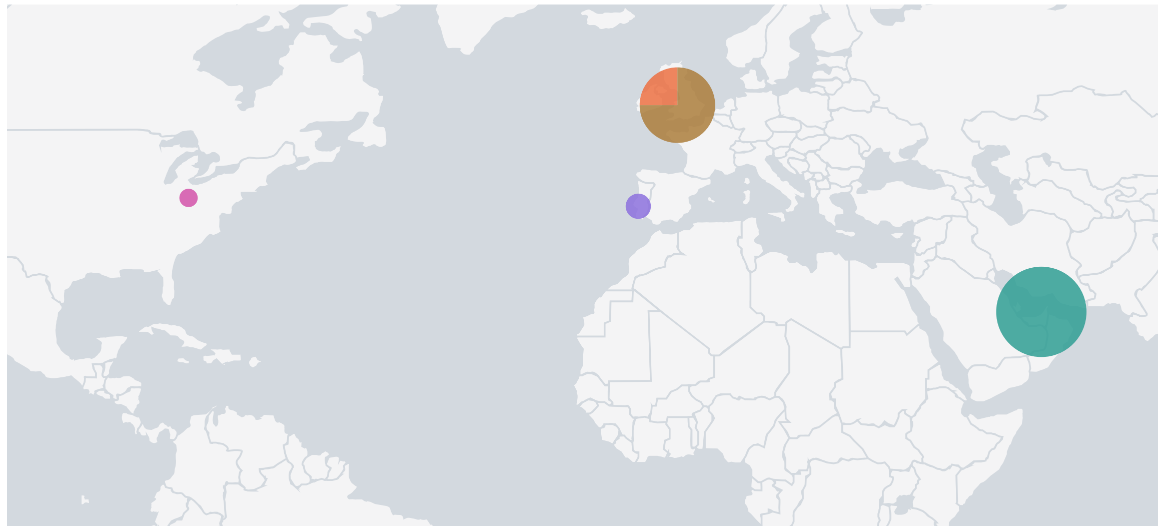
This query shows a map of geographic locations based on the IP address where an AWS long-term access key was used.

```
index IN (AWSCloudTrail) userIdentity.
type=IAMUser userIdentity.
accessKeyId=AKIAxxxxxxxxxxx
| stats count by sourceIPAddress
| iplocation sourceIPAddress
| geostats latfield=lat longfield=lon
count by City
```

Discussion

The **`iplocation`** command enriches the data by adding geographic information (such as latitude, longitude, and city) based on the `sourceIPAddress`. The **`geostats`** command then aggregates and visualizes this geographic data, allowing you to see event counts distributed by city, leveraging the latitude and longitude fields extracted by `iplocation`.

Cluster Map



Problem

You want to create a baseline, and identify when multiple first-time users connect to a new domain.

Solution

This detection first extracts and maintains a baseline list of all domains accessed in the environment. The data source would typically include proxy logs, DNS logs or endpoint logs with outbound HTTP or DNS traffic.

Baseline generation search (only needs to run once)

```
(index=proxy OR index=dns) (user="*"
user!="" user!="-" dest=* dest!=""
dest!="-") earliest="-30d"
| eval list="*"
| `ut_parse_extended(url,list)`
| eval domain=ut_domain
| fields - ut_* list
| stats earliest(_time) as earliest
latest(_time) as latest, count as
request_count by domain
| where latest > relative_time(now(),
"-30d")
| outputlookup user_domain_tracker.csv
```

***Special Ingredient:** the `ut_parse_extended` macro requires URL Toolbox, available on Splunkbase.*

Detection search

```
(index=proxy OR index=dns) (user="*"
user!="" user!="-" dest=* dest!=""
dest!="-") earliest="-1h"
| eval urlfield=coalesce(url, request,
domain,dest)
| eval list="*"
| `ut_parse_extended(urlfield,list)`
| eval domain=ut_domain
| fields - ut_* list
| where domain!="None"
| stats earliest(_time) as earliest
latest(_time) as latest, values(src) AS
src, values(dest) AS dest, values(source)
AS source, values(sourcetype) AS
sourcetype, values(index) AS index,
values(user) AS user by domain
| inputlookup append=t user_domain_
tracker.csv
| stats min(earliest) as earliest
max(latest) as latest, values(src) AS
src, values(dest) AS dest, values(source)
AS source, values(sourcetype) AS
sourcetype, values(index) AS index,
values(user) AS user by domain
| appendpipe
[| fields domain earliest latest
| stats min(earliest) as earliest
max(latest) as latest by domain
| where latest > relative_time(now(),
"-30d")
| outputlookup user_domain_tracker.csv
| where a=b
]
| eval maxlatest=now()
| eval isOutlier=if(earliest >= relative_
time(maxlatest, "-1h"), 1, 0)
| where isOutlier=1
| eventstats dc(user) AS users_for_
domain BY domain
| eval isOutlier=if(users_for_domain>1,
1, 0)
| where isOutlier=1
| convert timeformat="%Y-%m-%dT%H:%M:%S"
ctime(earliest), ctime(latest),
ctime(maxlatest)
| fields - src user
```

***Special Ingredient:** the `ut_parse_extended` macro requires URL Toolbox, available on Splunkbase.*

Discussion

This type of activity could be indicative of a spear phishing campaign, malware distribution, C2 communication, or a variety of other threats.

By using **appendpipe** with **outputlookup**, we can strip away fields that don't need to be saved in the baseline. This reduces the size and cardinality of the baseline significantly. Continuing with '| where a=b' closes off the branch for the search so we don't get duplicate events for the output. We have set a baseline max age of 30 days.

Problem

You want to create a baseline and detect when a first time seen user-agent string connects to only a single domain.

Solution

This detection uses URL Toolbox to extract the domain from the event and maintains a baseline of the domain count per user-agent. The rule is set to trigger when a completely new user-agent is detected that only has connected to a single domain in the time window of the alert (1h in this example).

Baseline generation search

The baseline search only needs to run once.

```
index=proxy (http_user_agent="*" http_
user_agent!="" http_user_agent!="-")
(dest=* OR url=* OR request=*)
earliest="-30d"
| eval urlfield=coalesce(url, request,
domain,dest)
| eval list="*"
| `ut_parse_extended(urlfield,list)`
| eval domain=ut_domain
| fields - ut_* list
| stats earliest(_time) as earliest
latest(_time) as latest, count AS
request_count, dc(domain) AS domain_
count by http_user_agent
| where latest > relative_time(now(),
"-30d")
| outputlookup useragent_domain_
tracker.csv
```

Detection search

```
index=proxy (http_user_agent="*" http_
user_agent!="" http_user_agent!="-")
(dest=* OR url=* OR request=*)
earliest="-1h"
| eval urlfield=coalesce(url, request,
domain,dest)
| eval list="*"
| `ut_parse_extended(urlfield,list)`
| eval domain=ut_domain
| fields - ut_* list
| where domain!="None"
| stats earliest(_time) as earliest
latest(_time) as latest, count
AS request_count, dc(domain) AS
domain_count, values(src) AS src,
values(user) AS user, values(dest)
AS dest, values(source) AS source,
values(sourcetype) AS sourcetype,
values(index) AS index by http_user_
agent
| inputlookup append=t useragent_
domain_tracker.csv
| stats min(earliest) as earliest
max(latest) as latest, sum(request_
count) AS request_count, sum(domain_
count) AS domain_count, values(src) AS
src, values(user) AS user, values(dest)
AS dest, values(source) AS source,
values(sourcetype) AS sourcetype,
values(index) AS index by http_user_
agent
| where latest > relative_time(now(),
"-30d")
| outputlookup useragent_domain_
tracker.csv
| eval maxlatest=now()
| eval isOutlier=if(earliest >= relative_
time(maxlatest, "-1h") AND (domain_
count=1), 1, 0)
| where isOutlier=1
| convert timeformat="%Y-%m-%dT%H:%M:%S"
ctime(earliest), ctime(latest),
ctime(maxlatest)
```

Discussion

Attackers often use custom or unusual user-agent strings when attempting to bypass traditional detection approaches. A new user-agent string connecting to a single domain may indicate a phishing attempt, malware communication, or another suspicious interaction.

Legitimate user-agent strings usually connect to multiple domains as part of normal browsing or API usage. A new user-agent string that only connects to a single domain within a specified time window could be a sign of Command and Control (C2) traffic, where the attacker is using a unique user-agent for communication with a specific domain under their control.

During the first run, legitimate new user-agents (e.g., newly deployed applications or updated browsers) may flag as outliers. These cases should be reviewed and added to the baseline as necessary to minimize false positives over time.

Problem

Create a baseline, and detect first-time seen egress communication from internal servers.

Solution

This search is designed to detect first-time egress communications from internal servers by comparing current traffic with a previously established baseline of source and destination IP pairs. The baseline is stored in a lookup file (egress_src_dest_tracker.csv), which is continuously updated to track ongoing communication patterns between internal servers and external destinations.

```
| tstats summariesonly=false allow_old_
summaries=true
    earliest(_time) as earliest
    latest(_time) as latest
    values(All_Traffic.action) as action
    values(All_Traffic.app) as app
    values(All_Traffic.dest_ip) as dest_
ip
    values(All_Traffic.dest_port) as
dest_port
    values(sourcetype) as sourcetype
count
    from datamodel=Network_Traffic
    where (NOT (All_Traffic.dest_
category="internal" OR All_Traffic.
dest_ip=10.0.0.0/8 OR All_Traffic.
dest_ip=172.16.0.0/12 OR All_Traffic.
dest_ip=192.168.0.0/16 OR All_Traffic.
dest_ip=100.64.0.0/10))
    by All_Traffic.src_ip All_Traffic.
dest_ip
| rename "All_Traffic.*" as *
| lookup egress_src_dest_tracker.
csv dest_ip src_ip OUTPUT earliest AS
previous_earliest latest AS previous_
latest
| eval earliest=min(earliest, previous_
earliest), latest=max(latest, previous_
latest)
| fields - previous_*
| appendpipe
    [
    | fields src_ip dest_ip latest
earliest
    | stats min(earliest) as earliest
max(latest) as latest by src_ip, dest_ip
    | inputlookup append=t egress_src_
dest_tracker.csv
    | stats min(earliest) as earliest
max(latest) as latest by src_ip, dest_ip
    | outputlookup egress_src_dest_
tracker.csv
        ]
| eventstats max(latest) as maxlatest
| eval comparisonTime="-1h@h"
| eval isOutlier=if(earliest >= relative_
time(maxlatest, comparisonTime), 1, 0)
| convert timeformat="%Y-%m-%dT%H:%M:%S"
ctime(earliest),ctime(latest)
,ctime(maxlatest)
| where isOutlier=1
```

Note: The first time you run this search, it will give an error unless the lookup “egress_src_dest_tracker.csv” is configured in the environment. You can easily get around that by running the search after removing this line:

```
| lookup egress_src_dest_tracker.
csv dest_ip src_ip OUTPUT earliest AS
previous_earliest latest AS previous_
latest
```

Afterwards, you can re-add this line to create a detection.

Discussion

First-time egress from internal servers could indicate potential compromise, data exfiltration, or other suspicious activity.

Initially, the search gathers all source (`src_ip`) and destination (`dest_ip`) IP pairs from egress traffic over a longer time window (e.g., 30 days) and stores these in a lookup file. This establishes a historical record of which internal servers have communicated with external IPs. In subsequent runs (e.g., every hour), the search compares current egress traffic against this baseline. It flags any new source-to-destination communication pairs that were not previously seen in the baseline as potential anomalies or first-time communications. After every detection run, the lookup file is updated with the latest observed communications, ensuring that new traffic patterns are added to the baseline for future comparisons.

Model-assisted methods

The Splunk Machine Learning Toolkit (MLTK), and the Splunk App for Data Science and Deep Learning (DSDL) offer advanced data science methods and algorithms for use on your Splunk data. MLTK uses the **`fit`** and **`apply`** commands to enable model training and inference, respectively. The Splunk App for DSDL also provides access to these algorithms through the same SPL, but within the added context of a Jupyter Lab environment for added extensibility to manipulate your data or import and connect with resources external to Splunk.

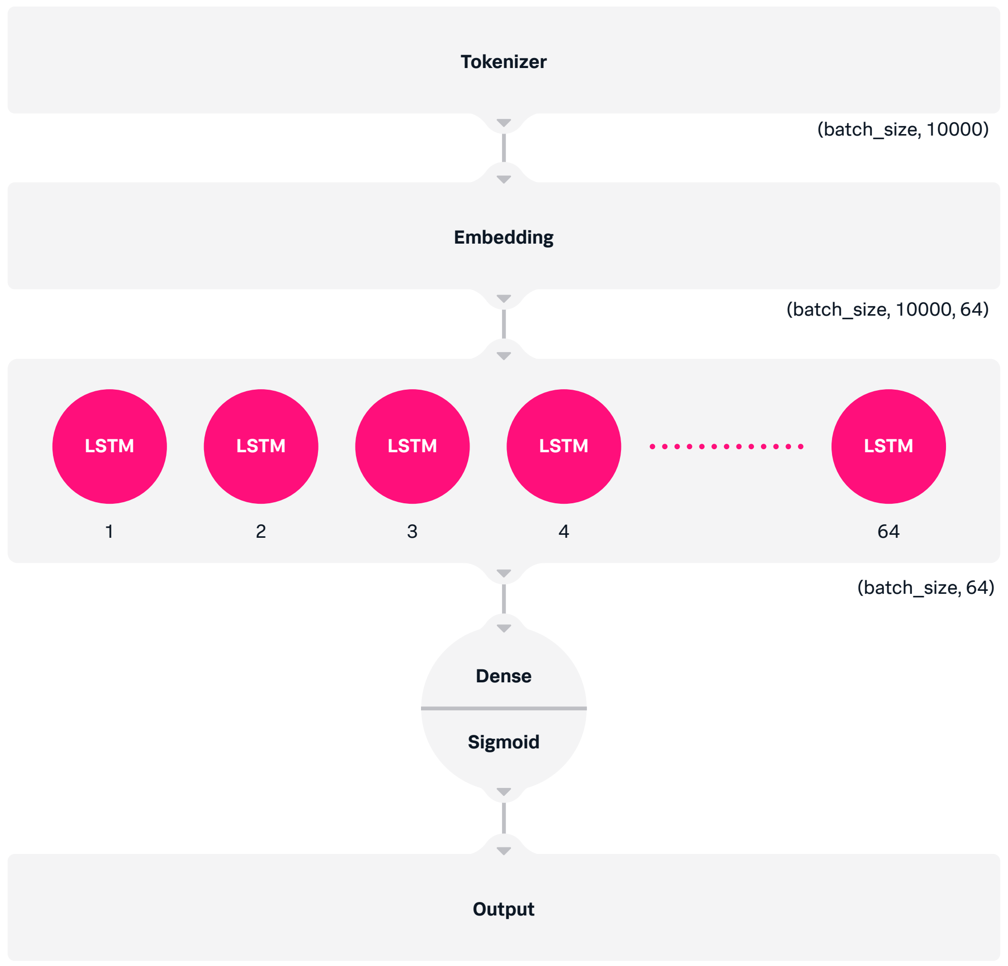
Classification Algorithms predict the value of a categorical field:

- **DecisionTreeClassifier**: A tree-based model that splits the data into branches to predict the label of a data point based on its features.
- **GradientBoostingClassifier**: An ensemble technique that combines multiple weak learners, typically decision trees, to create a strong predictive model.
- **LogisticRegression**: A statistical model that uses a logistic function to model binary dependent variables. It’s used for binary classification tasks.
    - Hunt web shells using a logistic regression classifier from the Splunk Threat Research Team, ShellSweepX!
- **RandomForestClassifier**: An ensemble learning method that uses multiple decision trees to improve the accuracy and robustness of predictions.
- **Deep Learning**: A method to train a neural network from scratch using the Splunk App for DSDL with the machine learning framework of your choice. Deep learning offers a powerful means to build classifiers when you have strong numerical features, or can encode the features used to distinguish between classes.

Hunters can use pre-trained models to bootstrap a M-ATH approach, e.g.:

- Detect Suspicious TXT Records Using Deep Learning encodes the features of DNS records using a word-embedding technique, followed by classification via a Long Short-Term Memory (LSTM) network. This model is available to be operationalized as a pre-trained model in the Splunk App for DSDL.

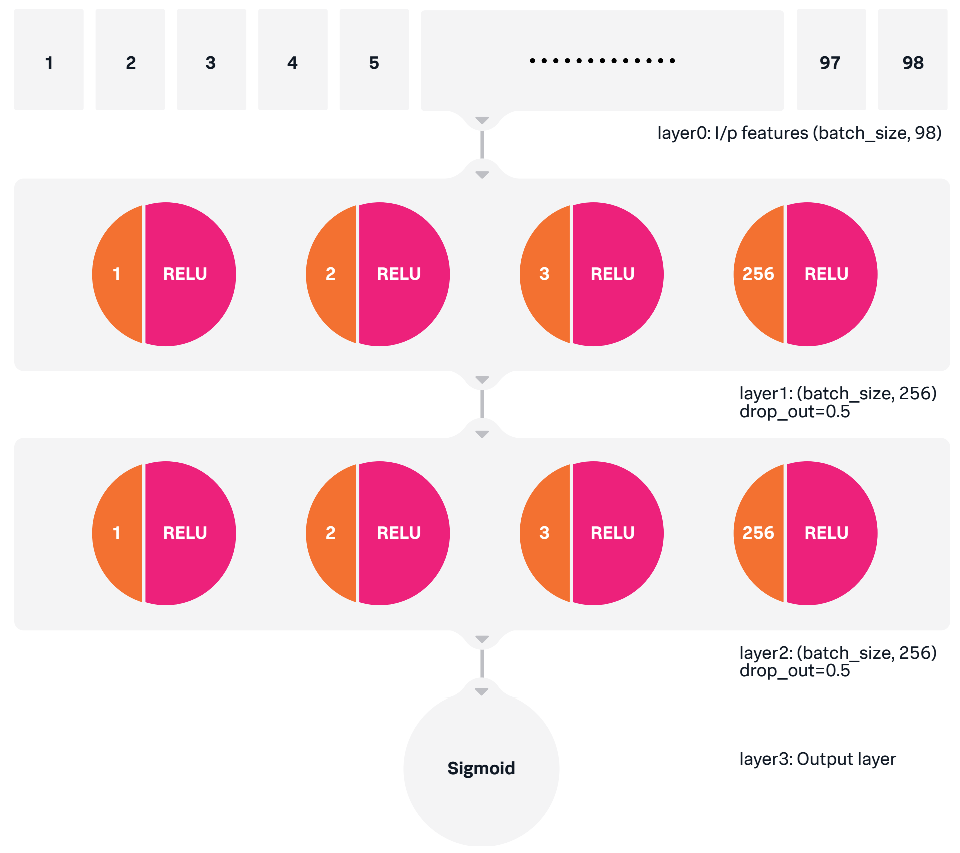
Splunk, 2023, Model Architecture



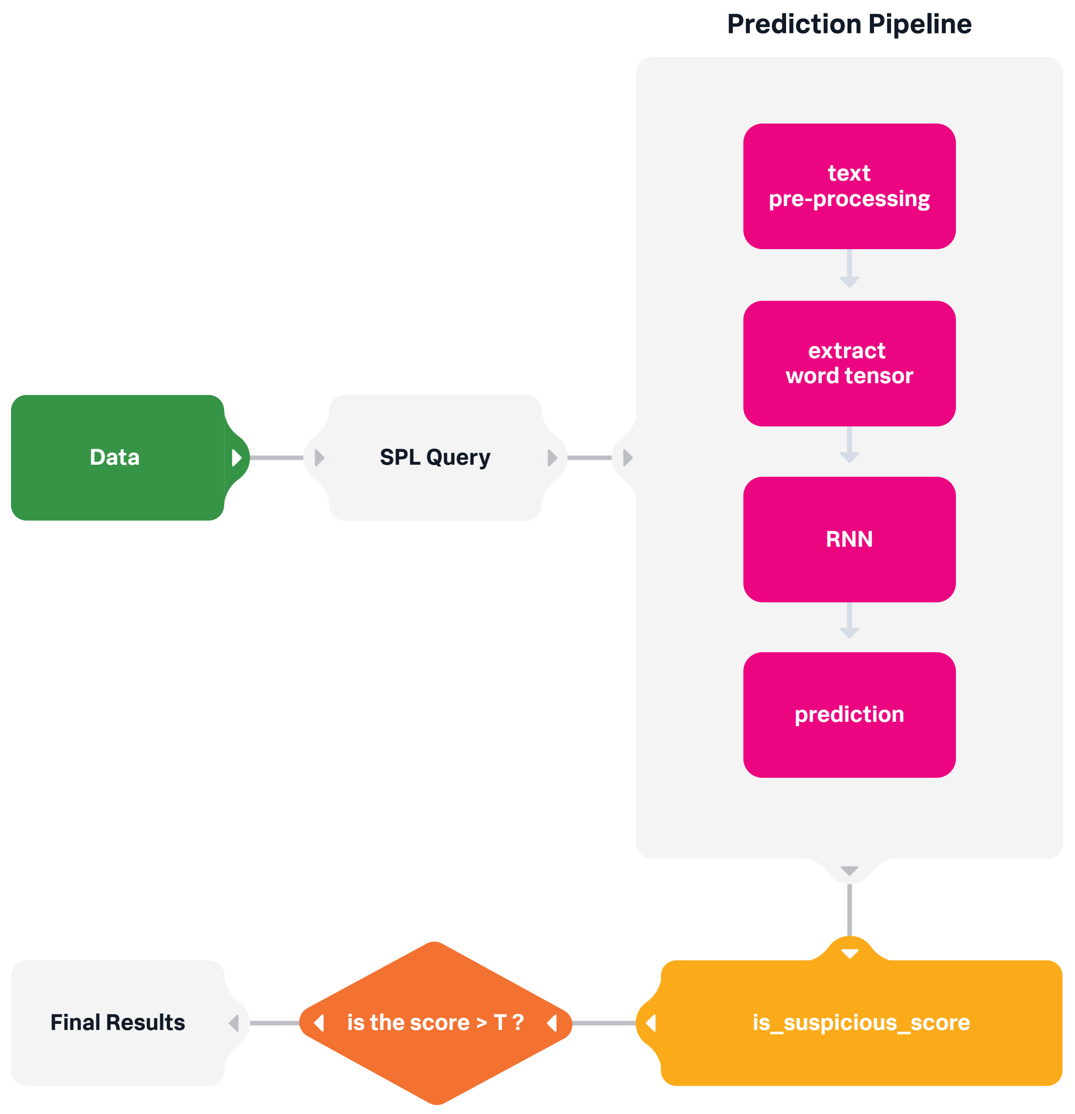
- Detecting DNS Data Exfiltration Using Deep Learning similarly extracts features from DNS request text via tokenization. This vector is concatenated with additional features like request length and entropy. The deep learning model used is a Multilayer Perceptron (MLP), a type of feedforward neural network with multiple dense layers, to detect DNS data exfiltration. This model is available to be operationalized as a pre-trained model in the Splunk App for DSDL.

- Detecting Suspicious Processes Using Recurrent Neural Networks encodes process names using a one-hot encoding technique at the character level. The encoded data is processed using a Recurrent Neural Network (RNN) to detect suspicious processes by distinguishing between randomly generated and legitimate process names. The RNN, due to its ability to capture sequence information, effectively learns patterns in the character sequences to classify them as benign or malicious. This model is available to be operationalized as a pre-trained model in the Splunk App for DSDL.

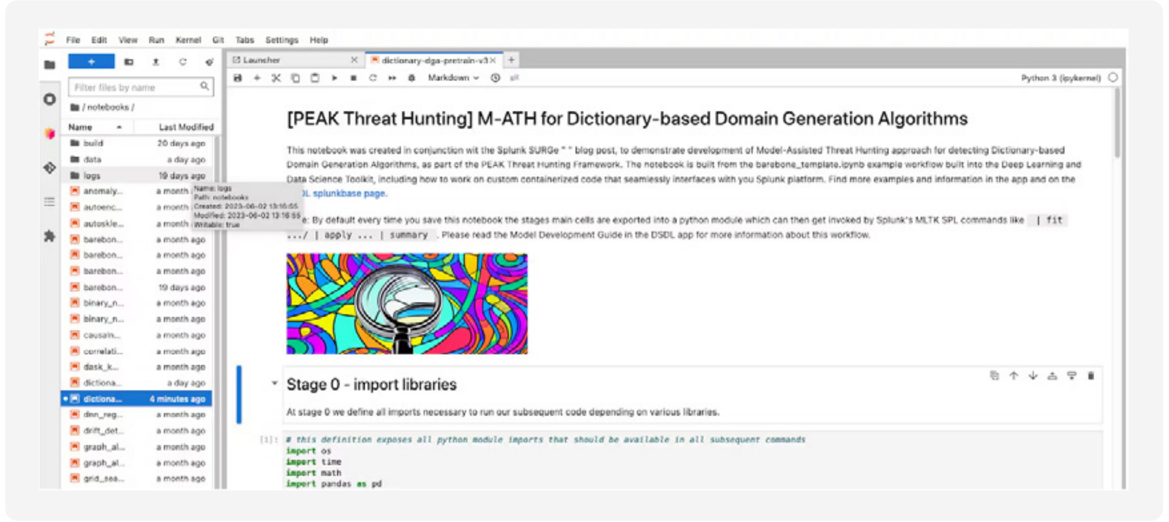
Splunk, 2023, Model Architecture



Splunk, 2023, Model Architecture



The Splunk App for DSDL


File Edit View Run Kernel Git Tabs Settings Help
Launcher
dictionary-dga-pretrain-v3
Markdown
Python 3 (ipykernel)
Name
Last Modified

[PEAK Threat Hunting] M-ATH for Dictionary-based Domain Generation Algorithms

This notebook was created in conjunction wit the Splunk SURGe " " blog post, to demonstrate development of Model-Assisted Threat Hunting approach for detecting Dictionary-based Domain Generation Algorithms, as part of the PEAK Threat Hunting Framework. The notebook is built from the barebone_template.ipynb example workflow built into the Deep Learning and Data Science Toolkit, including how to work on custom containerized code that seamlessly interfaces with you Splunk platform. Find more examples and information in the app and on the DSDL splunkbase page.

Note: By default every time you save this notebook the stages main cells are exported into a python module which can then get invoked by Splunk's MLTK SPL commands like `| fit .../ | apply ... | summary`. Please read the Model Development Guide in the DSDL app for more information about this workflow.

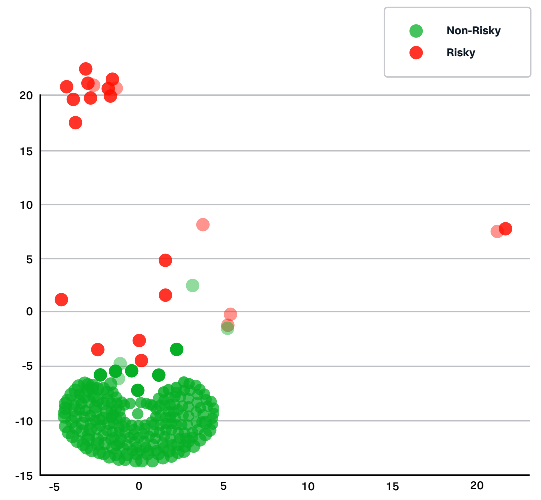
Stage 0 - import libraries

At stage 0 we define all imports necessary to run our subsequent code depending on various libraries.

```
# this definition exposes all python module imports that should be available in all subsequent commands
import os
import time
import math
import pandas as pd
```

- Detecting Domain Generation Algorithms (DGA) is also possible via deep learning. In two approaches from the Splunk platform, LSTM networks are trained to classify both traditional and dictionary-based DGAs. These models are available to be operationalized as pre-trained models in the Splunk App for DSDL.
- NLP-Based Risky SPL Detection facilitates threat hunting for suspicious queries in the Splunk platform. It utilizes a pre-trained deep learning model that combines an embedding layer with a transformer-based architecture. This approach helps in identifying potentially harmful queries by analyzing the syntax and structure of SPL commands, thus enhancing security monitoring. This model is available to be operationalized as a pre-trained model in the Splunk App for DSDL.

t-distributed Stochastic Neighbor Embedding, Risky vs. Non-Risky SPL



Special ingredients: Splunkbase add-ons

Looking to add some spice to your threat hunting recipe? Add-ons are the perfect seasoning to expand the functionality to your search, including support for security-specific use cases. These add-ons allow enrichment, correlation, and visualization of your data more effectively. To use them, you'll need to install additional apps on your instance of the Splunk platform through Splunkbase.

Working with URLs

Problem

Your hypothesis is that randomly-generated sub-domains will have high-entropy, and need to calculate it.

Solution

```
tag=dns
| `ut_parse(query)`
| lookup FP_entropy_domains domain AS
ut_domain
| search NOT FP_entropy=*
| `ut_shannon(ut_domain)`
| search ut_shannon&gt; 4.0
| stats count by query ut_shannon
```

***Special Ingredient:** the `ut_parse_extended` macro requires URL Toolbox, available on Splunkbase.*

Discussion

URL Toolbox is a set of building blocks for URL manipulation, enabling threat hunters to correctly parse URLs and complicated top-level domains (TLDs) using the Mozilla Public Suffix List. The toolbox add-on additionally supports Shannon entropy calculation, counting, suites, meaning ratio, and Bayesian analysis!

Problem

You want to hunt for data exfiltration attempts via DNScat2. Tools like DNScat2 facilitate exfiltration by generating high volumes of DNS requests for randomized subdomains. These subdomains embed exfiltrated data and target attacker-controlled domains, making detection challenging due to the use of legitimate DNS protocols for malicious purposes.

Solution

```
index=network sourcetype=stream:dns
| `ut_parse(query)`
| `ut_shannon(ut_subdomain)`
| eval sublen=len(ut_subdomain)
| stats count avg(ut_shannon) as
avg_sha avg(sublen) as avg_sublen
stdev(sublen) as stdev_sublen by ut_
domain
| search avg_sha > 3 avg_sublen > 20
stdev_sublen < 2
```

Discussion

- The **`ut_parse_extended(2)`** macro will give you a full extraction of the URL, including the domain, domain without TLD, subdomain, etc…
- The **`ut_levenshtein(2)`** macro measures distance in the number of substitutions made to transform one string to another.

Problem

You want to hunt for homoglyph or spoofed domains with low Levenshtein distance.

Solution

```
index=mail mail from
| stats count by Sender
| rex field = Sender "\@(?<domain_
detected>.*)"
| stats sum(count) as count by domain_
detected
| where domain_detected != "mycompany.
com"
| eval list = "mozilla"
| `ut_parse_extended(domain_detected,
list)`
| eval company_domain = "mycompany.com"
| `ut_levenshtein(ut_domain, company_
domain) `
| eval ut_levenshtein = min(ut_
levenshtein)
| where ut_levenshtein < 3
```

Discussion

Levenshtein distance is a string metric for measuring the difference between two sequences, for example, the Levenshtein distance between two words is the minimum number of single-character edits (i.e., insertions, deletions, or substitutions) required to change one word into the other. This hunt requires a target set of domains from which to measure this distance. By comparing incoming domains, URLs, or email addresses against a known list of trusted domains using the Levenshtein distance algorithm, we can identify subtle variations that indicate potential spoofing or homoglyph attacks.

Visualizing process relationships

Problem

You want to hunt for processes launching from suspicious locations (i.e., temp directories, startup folders, etc).

Solution

PSTree for Splunk enables a custom search command in the Splunk platform to reconstruct a pstree from Sysmon process creation events (EventCode 1).

```
index=sysmon
source="xmlwineventlog:microsoft-
windows-sysmon/operational" EventCode=1
user=<user>
| rex field = ParentImage "\
x5c(?<ParentName>[^\x5c]+)$"
| rex field = Image "\
x5c(?<ProcessName>[^\x5c]+)$"
| eval parent = ParentName."
(".ParentProcessId.")"
| eval child = ProcessName."
(".ProcessId.")"
| eval detail = strftime(_time,"%Y-%m-%d
%H:%M:%S")." ".CommandLine
| pstree child = child parent = parent
detail = detail spaces = 50
| search tree = *Salaries.xls*
| table tree
```

***Special Ingredient:** This recipe requires the PSTree for Splunk app, available on Splunkbase.*

Example PSTree output

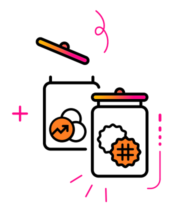
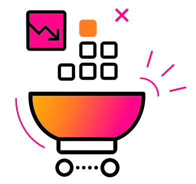
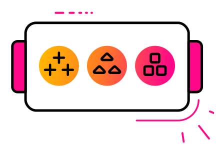
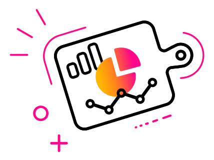
```
WmiPrvSE.exe (2240)
|--- powershell.exe (4976)
.. |--- eventvwr.exe (3800)
.. .. |--- powershell.exe (4468)
.. .. .. |--- powershell.exe (3712)
.. .. .. .. |--- ftp.exe (4540)
.. .. .. .. |--- schtasks.exe (3360)
.. .. .. .. |--- netch.exe (4456)
.. .. .. .. |--- csc.exe (4360)
.. .. .. .. |--- cvtres.exe (4808)
.. .. .. .. |--- net.exe (2128)
.. .. .. .. |--- net1.exe (4696)
.. .. .. .. |--- whoami.exe (4668)
.. |--- whoami.exe (4396)
.. |--- eventvwr.exe (3816)
.. |--- whoami.exe (1512)
```

Discussion

PSTree reconstructs process trees from Sysmon EventCode 1 data, which tracks process creation events. With this output, we can hunt suspicious processes launching from unusual directories, such as temporary folders or startup locations, that are often used by malware.

The query first extracts the parent process name (`ParentName`) and the child process name (`ProcessName`) from their respective paths using `rex` commands. It then combines these with their process IDs to create descriptive `parent` and `child` fields, making the process relationships clear. The `detail` field is created to include the timestamp and command line, providing additional context for each process.

Quick reference chart

Method	Summary	SPL Functions	Algorithms and Visualizations
Searching and Filtering	Your go-to, your Chef's knife, Searching and Filtering is for slicing, dicing, and extracting the data you need.	• metadata • datamodel • xyseries • untable • eval • logical operators • relational operators • lookup • regex • rex	
Sorting and Stacking	Interested in the highest volume or the rarest values? Try Sorting and Stacking to see what piles up or sifts through.	• stats • top • rare • sort • eventstats • streamstats	• Bar and Column Charts • Line and Area Charts
Grouping 	Looking for complementary flavors, like lateral movement + privilege escalation? Grouping will connect actions and events into groups with these functions.	• bin • stats – `values` – `dc` – `count, avg, min, max...` • join • transaction	
Forecasting and Anomaly Detection 	Searching for something out-of-the ordinary? Anomaly Detection recipes will help you find events that stand out.	• fillnull • timechart • eventstats • eval • anomalydetection	• DensityFunction • OneClassSVM • ARIMA • StateSpace Forecast
Clustering 	Wondering which ingredients would stick together? Use Clustering to automatically measure associations between events based on the attributes you define.	• stats • cluster • fields • kmeans • xyseries	• K-Means • PCA • TFIDF • 3D Scatter Plot
Exploratory Data Analysis and Visualization 	Know thyself — but also, know thy data. EDA and Visualization techniques use statistics and graphical means to establish a baseline understanding of our data.	• fieldsummary • transpose • chart • bin • metadata • timechart • sparkline	• Scatter Plot • Box Plot
Combined Methods	A master class in culinary data science — apply these signature methods for testing multi-stage hypotheses, training classification models, or adding new ingredients to your repertoire!	• iplocation • geostats • appendpipe • outputlookup	• DecisionTree Classifier • GradientBoosting Classifier • Logistic Regression • RandomForest Classifier • URL Toolbox • PSTree

More Splunkbase favorites

- Event Timeline Viz - https://splunkbase.splunk.com/app/4370 >
- Configuration Manager (Conf Manager) - https://splunkbase.splunk.com/app/6895 >
- Splunk App for Behavioral Profiling - https://splunkbase.splunk.com/app/6980 >
- Tuning Framework for Splunk - https://splunkbase.splunk.com/app/6943 >
- Splunk Enterprise Security - https://splunkbase.splunk.com/app/263 >
- SA-Investigator for Enterprise Security - https://splunkbase.splunk.com/app/3749 >
- SA-DetectionInsights - https://splunkbase.splunk.com/app/7066 >

Explore the Threat Hunter's Cookbook search collection:

- Insights Suite for Splunk (IS4S) - https://splunkbase.splunk.com/app/7186 >

SURGe by Splunk

As always, security at Splunk is a family business. Special thanks to our contributors, including:

- **David Bianco**
- **Johan Bjerke**
- **Robin Burkett**
- **James Callahan**
- **Tyne Darke**
- **Derek Du**
- **Jordan Fuentes**
- **Megan Jooste**
- **Audra Streetman**
- **Jesse Trucks**

25_CMP_ebook_threat-hunters-cookbook_digital_v4

ISBN 979-8-218-69420-3

9 798218 694203

splunk> a CISCO company